UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Titulación:

INGENIERÍA EN AUTOMÁTICA Y ELECTRÓNICA INDUSTRIAL

Alumno:

USMAN ALI

Título del TFG:

PROYECTO DE DESARROLLO DE UNA APLICACIÓN DE GESTIÓN DE UN SERVIDOR OPC PARA AUTOMATIZACIÓN INDUSTRIAL.

Director del TFG:

RAMON SARRATE ESTRUCH

Convocatoria de entrega del proyecto:

Junio, 2020

Contenido del documento:

MEMORIA

Declaración de honor

Yo USMAN ALI, estudiante de grado en INGENIERÍA EN AUTOMÁTICA Y ELECTRÓNICA INDUSTRIAL, de la ESEIAAT - Escuela Superior de Ingenierías Industrial, Aeroespacial y Audiovisual de Terrassa.

DECLARO QUE:

La memoria del Trabajo Final de Grado denominado, PROYECTO DE DESARROLLO DE UNA APLICACIÓN DE GESTIÓN DE UN SERVIDOR OPC PARA AUTOMATIZACIÓN INDUSTRIAL ha sido desarrollado respetando la propiedad intelectual de otras personas, citando las fuentes bibliográficas utilizadas en la redacción de dicho trabajo.

En virtud de esta declaración afirmo que este trabajo es inédito y de mi autoría, por lo que me responsabilizo del contenido de la Memoria del Trabajo Final de Grado, y asumo que cualquier infracción de esta declaración me deja sometido a acciones disciplinarias previstas por la Universidad Politécnica de Cataluña - BarcelonaTECH.

Usman Ali 27 de junio de 2020

Agradecimientos

A Fernando Campos, Mikel Carril, Ramon Sarrate y Xavier López que, gracias a sus consejos, el tiempo dedicado y las correcciones se ha logrado un trabajo de gran calidad, que me ha dado la oportunidad de aprender algo nuevo y ampliar mis conocimientos adquiridos durante la carrera.

A mi familia por el apoyo y la confianza que han sido la clave para seguir trabajando día tras día.

A todos los amigos de la universidad en especial Anna, Javier y Gerard por hacer que esta etapa haya sido tan alegre y divertida.

Gracias.

Resumen

El presente documento tiene como finalidad recoger el Trabajo Final de Grado, realizado por el alumno Usman Ali de la ESEIAAT - Escuela Superior de Ingenierías Industrial, Aeroespacial y Audiovisual de Terrassa. La misión de este proyecto es desarrollar una aplicación para la configuración y gestión de un servidor OPC alojado en un sistema Linux. La aplicación se comunicará con un servidor OPC Thingworx Kepware Edge a través de una API REST, ofreciendo una interfaz de configuración amigable.

Para comprender el funcionamiento del servidor OPC y su API REST de configuración, se ha apoyado en una serie de pruebas de configuración y comunicación, instalando el ThingWorx Kepware Edge en un ordenador con sistema operativo Linux.

Se ha desarrollado una aplicación SCADA que obtiene los datos de una RTU, usando servidor OPC como intermediario. Tras construir ambas aplicaciones, se ha utilizado la aplicación SCADA para demostrar el correcto funcionamiento de la aplicación de configuración desarrollada.

Abstract

The purpose of this document is to collect the Final Degree Project, carried out by the student Usman Ali of ESEIAAT - Escuela Superior de Ingenierías Industrial, Aeroespacial y Audiovisual de Terrassa. The objective of this project is to develop an application for the configuration and management of an OPC server hosted on a Linux system. The application will communicate with a Thingworx Kepware Edge OPC server via a REST API, providing a user-friendly configuration interface.

To understand how the OPC server works and its configuration through REST API, it has relied on a series of configuration and communication tests, installing the ThingWorx Kepware Edge on a personal computer running Linux.

A SCADA application has been developed that obtains the data from an RTU, using OPC server as an intermediary. After developing both the applications, the SCADA application has been used to demonstrate the correct operation of the developed configuration application.

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

Cada vez hay más empresas que están apostando por actualizar las fábricas tradicionales, para que sean más inteligentes y eficientes. De esta forma las instalaciones son dotadas con la capacidad de autoadaptación ante los medios de producción y aprovechar al máximo todos los recursos disponibles.

Todo esto es posible gracias al concepto de Industria 4.0. Que consiste en la digitalización de todos los procesos de la fábrica y todos los servicios relacionados con la empresa. El mercado actual ha desarrollado muchos productos que permiten cumplir con esta misión. Ya sean sensores inteligentes, PLC's con protocolos de comunicación industriales más robustos y avanzados, sistemas SCADA que ya no tan solo sirven para supervisar un proceso sino también para generar informes, almacenar datos en bases de datos, etc. También se han introducido soluciones que se integran con la fábrica para la planificación de recursos empresariales (ERPs), integración con las plataformas IoT, sistemas MES (Manufacturing Execution System) dedicados para controlar la producción, monitorización y documentación de la gestión de la planta.

Todas estas tecnologías obviamente tienen que estar interconectadas para el intercambio de la información. Para ello hace falta un protocolo de comunicación que sea un estándar, fácil de integrar, que permita una comunicación segura y que sea interoperable a todos los niveles.

OPC con su especificación actual OPC UA es uno de los protocolos de comunicación estándar que está ganando el terreno de la interoperabilidad, ya que cumple con todas las necesidades mencionadas anteriormente. Todas las soluciones anteriores están implementando OPC UA como protocolo de comunicación para la obtención de los datos desde la fábrica. Su arquitectura cliente-servidor es utilizada por soluciones basadas en plataformas como Linux, Windows, MacOS , PLC's con sistema operativo propios, dispositivos inteligentes (IoT) y herramientas basadas en la nube como son Azure IoT Hub.

La ventaja principal de usar un estándar de comunicación que no sea propiedad de ningún fabricante es que, permite integrar dispositivos de diferentes fabricantes en una misma planta. De esta forma si se desea substituir un elemento por otro con mejores prestaciones, no se tiene que cambiar todo el sistema ya montado.

Cambiar todos los dispositivos ya instalados en una fábrica, por unos compatibles con la tecnología OPC no tiene mucho sentido, ya que se necesitaría una inversión considerable de dinero y tiempo. Pero existe una solución más económica para lograr la integración de la tecnología OPC que es, usar un servidor OPC que hace de intermediario entre los dispositivos de campo y los sistemas superiores. Los dispositivos de campo comunican e intercambian datos con el servidor OPC utilizando el protocolo de comunicación que tengan de forma nativa (ModBus, Profibus, Profinet, PowerLink, etc.). El servidor OPC se encarga de servir estos datos a los sistemas de nivel superior, para su posterior explotación.

KEPServerEX es una plataforma de comunicaciones industriales que fue desarrollada por Kepware y es distribuida por Logitek a nivel de España y Portugal. Se ha convertido en líder en el sector de la automatización industrial, con centenares de plantas en España y Portugal que implementan esta solución. Empresas como Aena, Damm, San Miguel, Mercadona,

Ayuntamiento de Barcelona, Coca-Cola y muchas más, ya han implementado esta solución en sus instalaciones.

Es capaz de intercambiar datos prácticamente con la mayoría de los dispositivos industriales (PLC, RTU, BBDD, etc.). Centraliza todos estos datos y los sirve a los sistemas superiores vía OPC y otros protocolos industriales más avanzados como, por ejemplo, MQTT,REST,SNMP o ODBC.

KEPServerEX es una versión del servidor para los sistemas operativos de Windows. Pero Kepware recientemente ha liberado una versión llamada ThingWorx Kepware Edge, para sistemas operativos Linux.

Esta nueva versión tiene la capacidad de correr en ordenadores que no requieren gran capacidad de computación. Se puede instalar en SBC (Single Board Computers), ordenadores de placa única.

Las dos versiones disponen una API REST de configuración , que permite configurar el servidor de forma remota. Es decir, se puede configurar el servidor desde cualquier máquina que tenga acceso a la máquina donde está instalado el servidor, simplemente enviando peticiones HTTP.

Lamentablemente, ThingWorx Kepware Edge actualmente no dispone de una interfaz gráfica de configuración como KEPServerEX. Por lo tanto, esta versión del servidor OPC, solamente permite configuración mediante la API REST de configuración.

1.1. Objetivo

El objetivo de este proyecto es desarrollar una aplicación de configuración para el servidor ThingWorx Kepware Edge. Se utilizará la interfaz API REST para comunicar la aplicación de configuración con el servidor OPC.

El proyecto se enfocará en ThingWorx Kepware Edge y la aplicación a crear será 100 % pensada para esta versión de servidor OPC. Pero dado que la interfaz API REST de configuración es “idéntica” para las dos versiones, la aplicación también permitirá configurar KEPServerEX, pero con ciertas limitaciones.

Para estudiar la API REST de configuración se pretende realizar pruebas de comunicación, instalando ThingWorx Kepware Edge en un ordenador con sistema operativo Linux. Como fuente de datos se utilizará una RTU (Unidad Terminal Remota), que comunicará con el servidor OPC vía ModBus. Se utilizarán varios clientes OPC UA que comunicarán con el servidor OPC, para leer/escribir los datos provenientes de la RTU.

Para demostrar el correcto funcionamiento de la aplicación de configuración, se desarrollará una segunda aplicación SCADA, que actuará como cliente OPC UA.

1.2. ¿Qué aportará esta aplicación de configuración?

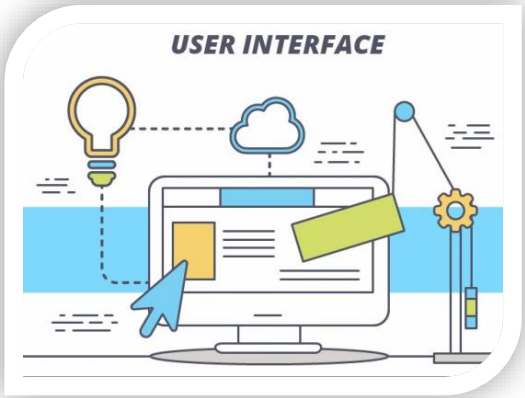


Figura 1-1. Concepto de una interfaz gráfica de usuario. Fuente: freepik.com

El objetivo principal de crear una aplicación es permitir al usuario final configurar la plataforma ThingWorx Kepware Edge de una **forma más fácil y amigable (**ver **Figura 1-1).** De esta forma el usuario no está obligado a tener conocimientos específicos del funcionamiento de la API REST.

Pero supongamos un caso real como el de una empresa como Aena, que puede tener muchos servidores OPC desplegados en sus numerosas instalaciones.

Para configurar cada uno de estos servidores OPC, el usuario debe de acceder físicamente o de forma remota, **a cada ordenador** donde está instalado el servidor para poder configurarlo.

Aquí es donde entra en acción la aplicación de configuración. Ya que, **utilizando esta aplicación,** el usuario, **desde un único ordenador** (conectado con todos los servidores OPC), puede configurar todos los servidores OPC de forma remota.

1.3. Alcance

Para el éxito de este proyecto se han definido las siguientes tareas:

- Búsqueda de información y documentación necesaria para comprender el funcionamiento de la API REST de la plataforma de comunicaciones industriales ThingWorx Kepware Edge.
- Búsqueda de información para la instalación del sistema operativo Ubuntu, en un ordenador portátil.
- Instalación del sistema operativo Linux junto con el servidor OPC en el ordenador portátil.
- Familiarización con las peticiones HTTP y la estructura de datos en formato de texto tipo JSON.
- Programación de la RTU para leer los datos desde un módulo de simulación de entradas y salidas.
- Configuración del servidor OPC, para la comunicación con la RTU.
- Curso online para aprender a programar con C#.

    - Maestro en C# en tiempo récord.
- Instalación de Visual Studio, como el entorno de programación.
- Estudio sobre el diseño de la interfaz gráfica de la aplicación de configuración.
- Programación de la aplicación de configuración.
- Testing de la aplicación de configuración.
- Búsqueda de información y documentación para instalar InTouch Edge.
- Creación y programación de la aplicación SCADA.
- Demostración del correcto funcionamiento de la aplicación de configuración desarrollada, configurando en Servidor OPC para la visualización de los datos provenientes de la RTU, en la aplicación de SCADA.

1.4. Material necesario

- Ordenador con sistema operativo Linux.
- Aplicación ThingWorx Edge.
- RTU para la generación de datos a enviar al servidor OPC.
- La unidad terminal remota (RTU) MS de LKRemote + Módulo de simulación
- Fuente de alimentación de 24 V, para la RTU.
- Cables Ethernet para la comunicación entre el PLC y el ordenador donde estará alojado el servidor OPC.
- Ordenador con sistema operativo Windows, para instalación del entorno de programación.
- Softwares necesarios:
    - TwinSoft
    - Visual Studio
    - Postman
    - KEPServerEX
    - UAExpert
    - InTouch Edge

2. Introducción a los servidores OPC en el entorno de automatización industrial

2.1. Industria 4.0

Figura 2-1. Industria 4.0. Fuente: www.atriainnovation.com

La industria 4.0 (ver Figura 2-1) es un nuevo modelo para organizar los sistemas de producción, quedando todos interconectados. De manera que las máquinas están conectadas con los sistemas, y éstos a su vez, con las personas. Este cambio aporta una mejora considerable en la eficiencia para la gestión de una fábrica.

Apoyado por el gobierno alemán, en Europa también es habitual referirse a este concepto con términos como “Fabrica inteligente” o “internet industrial”. En definitiva, es una aplicación directa del modelo “Internet de las cosas” (IoT) en la industria. Todos estos términos tienen en común el reconocimiento de que los procesos de fabricación se encuentran en un proceso de transformación digital, una "revolución industrial" producida por el avance de las tecnologías de la información y particularmente de la informática y el software.

2.1.1. La cuarta revolución industrial

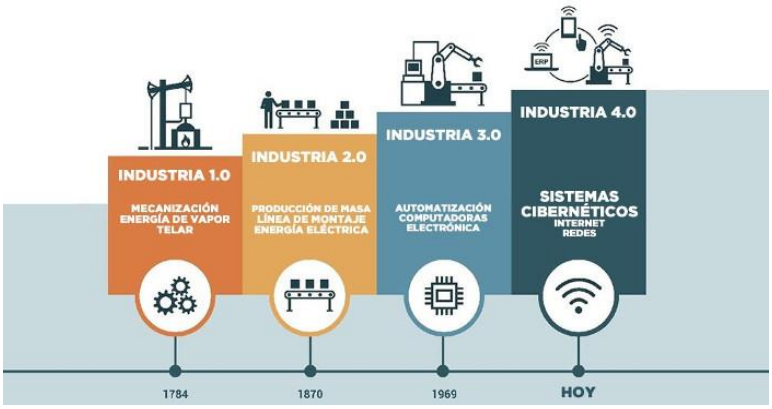


Figura 2-2. Evolución de la industria. Fuente:www.lupeon.com

Entre los siglos XVIII y XIX se da lugar la primera revolución industrial (ver Figura 2-2), donde se mecanizaron los procesos de producción, transformando la economía agraria y artesanal en otra liderada por la industria.

El segundo gran cambio en la industria se produjo en el siglo XX. Donde se introduce la producción en serie, con la aparición de fábricas y líneas de montaje que permitieron fabricar productos para el gran consumo.

El final de siglo XX trae una nueva transformación en la industria, que aprovecha la electrónica y la informática para automatizar las líneas de producción y que las máquinas reemplazaran a las personas en tareas repetitivas.

Tras dos décadas de desarrollo y grandes avances en la tecnología de internet han producido un impacto importante en la economía y en la industria. La combinación de las tecnologías de la información con la sensórica y la robótica están transformando la internet tradicional en internet de las cosas (IoT).

2.1.2. Internet of things (IoT)

Figura 2-3. Internet of Things. Fuente: www.muycanal.com

Internet de las cosas es la idea principal que sostiene la **industria 4.0 (**ver **Figura 2-3)**. Nació para establecer una comunicación inteligente entre las cosas y lleva más de 10 años revolucionando el mundo. Esta idea de conectividad global entre las cosas es aprovechada por numerosos sectores de la industria actual. Por ejemplo, en el ámbito de la medicina y la salud, a través de sistemas que permiten el control remoto de pacientes y relojes inteligentes que son capaz de registrar los kilómetros recorridos por una persona.

2.1.3. Internet Industrial de las Cosas (IIoT)

Figura 2-4. Internet industrial de las cosas (IIoT). Fuente: vestertraining.com

La Internet Industrial de las Cosas (IIoT) es el uso de tecnologías de internet de las cosas (IoT) en el ámbito industrial.

IIoT incorpora el aprendizaje de máquina y la tecnología de grandes volúmenes de datos (big data), aprovechando los datos de sensores, comunicación entre máquinas (M2M) y las tecnologías de la automatización. Tras recopilar información, estos datos pueden servir a una empresa para captar las ineficiencias y anticiparse a los problemas de la producción, ahorrando así, tiempo y dinero. IIoT tiene un gran potencial para el control de calidad, las prácticas sostenibles y verdes, la trazabilidad de la cadena de suministro y la eficiencia general de la cadena de suministro.

Uno de los grandes retos (ver Figura 2-5) que afronta la Internet Industrial de las Cosas es la interoperabilidad entre dispositivos y máquinas que utilizan diferentes protocolos de comunicación con diversas arquitecturas.

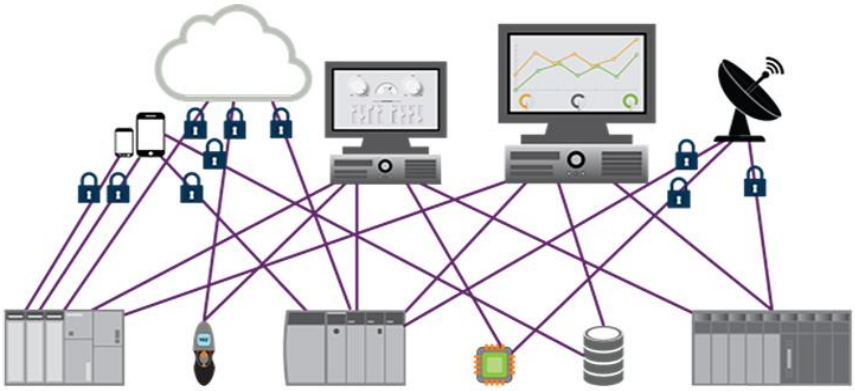


Figura 2-5. Comunicaciones industriales sin un centralizador de datos. Fuente: www.kepserverexopc.com

Hoy en día es muy común encontrar dispositivos de diferentes fabricantes en una misma planta. A medida que la industria va evolucionando, en estos dispositivos se están incorporando diferentes protocolos de comunicación estándares, como por ejemplo OPC DA, OPC UA, MQTT, REST, SNMP, etc.

Pero también es posible que nos enfrentemos con dispositivos, en los que el fabricante utilice su propio protocolo de comunicación que no suele ser compatible con el de otros fabricantes.

Por lo tanto, la misión de interconectar y compartir información entre todos los dispositivos de nuestra planta se vuelve prácticamente imposible.

Una solución sencilla y práctica para este tipo de problemas es (ver Figura 2-6), centralizar todos los datos en un único servidor de datos. De manera que todos los dispositivos de la planta están conectados, a este servidor.

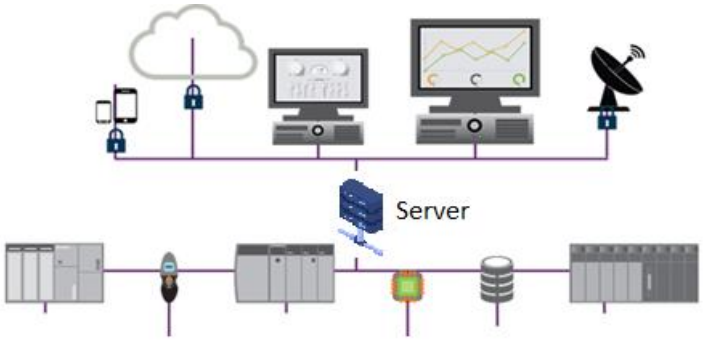


Figura 2-6. Arquitectura con un servidor como intermediario. Fuente: www.kepserverexopc.com

Este servidor debe de tener la capacidad de comunicarse prácticamente con todos los protocolos de campo existentes en el mercado, y a su vez servir estos datos a los sistemas superiores (SCADAs, EMS, GMAOs, plataformas IoT, …).

Por ejemplo, intercambiar los datos con un sistema SCADA, almacenar los datos en una base de datos, para que a posteriori se pueda transformar esta información en conocimiento útil para la empresa.

La industria actual está apostando por un estándar de comunicación llamado OPC. Por lo tanto, en el mercado actual ya existen varios servidores OPC que solucionan el problema de interoperabilidad.

2.2. ¿Qué es OLE for Process Control (OPC)?

Según la **OPC Foundation**, OPC es un estándar de interoperabilidad para el intercambio de datos en una forma segura y confiable en el ámbito de la automatización industrial. Garantiza el continuo flujo de información entre los dispositivos de múltiples fabricantes sin depender de la plataforma. La fundación OPC es el responsable del desarrollo y mantenimiento de este estándar.

OPC fue lanzado por primera vez en 1996 y su propósito era abstraer protocolos específicos de PLC (como ModBus, Profibus, etc.) en una interfaz estandarizada que permitiera a los sistemas HMI/SCADA interactuar con un “intermediario” genérico.

Como resultado, surgió toda una industria artesanal de productos que permitía a los usuarios finales implementar sistemas utilizando los diferentes dispositivos, todos interactuando sin problemas a través de OPC.

El OPC **(OLE for Process Control) clásico** estaba restringido al sistema operativo Windows, ya que estaba basado en OLE (object linking and embedding) para Control de Procesos, que es una tecnología de Microsoft. A posteriori estas especificaciones, se han adaptado de una forma más generalizada en múltiples industrias, incluidas la fabricación, la automatización de edificios, el petróleo y el gas, las energías renovables y los servicios públicos, etc. Por lo tanto, la definición **actual de OPC** según la OPC Foundation es **Open Platform Communications**.

La comunicación OPC se realiza a través de una arquitectura Cliente-Servidor (ver Figura 2-7). El servidor OPC es la fuente de datos (como un dispositivo hardware a nivel de planta) y cualquier

aplicación basada en OPC puede acceder a dicho servidor para leer/escribir cualquier variable que ofrezca el servidor.

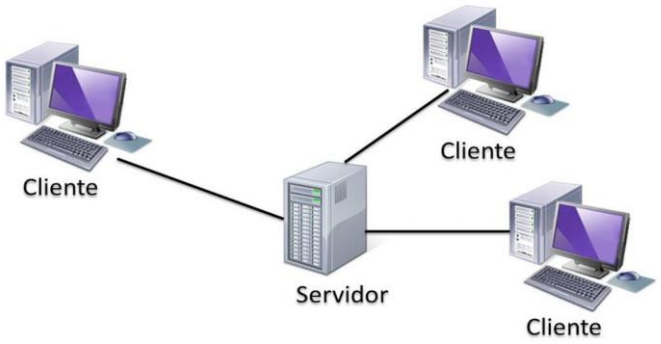


Figura 2-7. Arquitectura cliente-servidor. Fuente: entreunosyceros.net

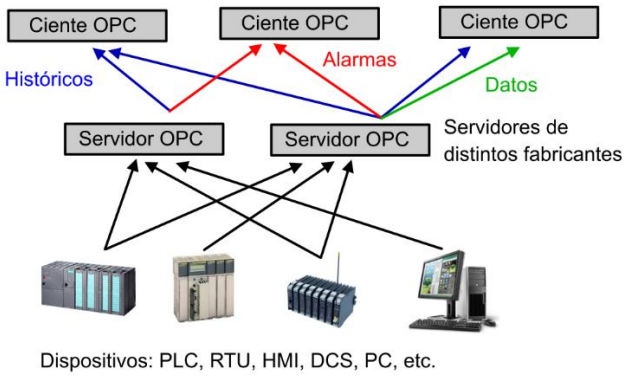


Figura 2-8. comunicación entre los dispositivos de campo y los clientes finales a través del servidor OPC. Fuente: josecasares.com

Un servidor OPC es una aplicación software basada en una o más especificaciones definidas por la OPC Foundation. El servidor OPC hace de interfaz "intermediario" que comunica con una o más fuentes de datos utilizando sus protocolos nativos y servir estos datos a un sistema superior (ver **Figura 2-8**). Las comunicaciones entre el Cliente OPC y el Servidor OPC son bidireccionales, lo que significa que los clientes pueden leer y escribir en los dispositivos a través del servidor OPC.

Tipos de especificaciones definidas por la OPC Foundation:

- OPC Data Access (DA): diseñado para la transferencia de datos en tiempo real. Depende de DCOM (un componente de Windows).
- OPC Alarms and Events (AE): transfiere Alarmas y Eventos desde el dispositivo hacia el Cliente OPC.
- OPC Historical Data Access (HDA): basado en la especificación de Acceso a Datos Historizados que provee al Cliente OPC HDA de datos históricos.
- OPC XML Data Access (XML-DA).

- OPC Data Exchange (DX)
- OPC Security
- OPC Unified Architecture (UA): basado en la especificación de Arquitectura Unificada – basado en el set más nuevo y avanzado de la OPC Foundation, permite a los Servidores OPC trabajar con cualquier tipo de datos.

  Para que un cliente pueda comunicarse con el servidor OPC-UA, este necesita un OPC-UA Endpoint (conexión). Este Endpoint contiene la dirección IP y el puerto del servidor. A parte del Endpoint el cliente también debe tener el usuario y la contraseña para la autentificación dependiendo del nivel de seguridad que tiene configurado este.

  Un ejemplo de OPC-UA Endpoint:

  ```
  opc.tcp://<IP del servidor>:49330(por defecto)
  ```

  Nota: Es muy importante no confundir el OPC-UA Endpoint con la dirección URI que también se denomina Endpoint.

En este proyecto se utilizará un servidor OPC desarrollado por Kepware. Tiene dos versiones que son, el KEPServerEX para sistemas operativos Windows, y el ThingWorx Kepware Edge para sistema operativos basados en Linux.

2.3. KEPServerEX

KEPServerEX es un servidor basado en software para sistemas operativos Windows, que está diseñado para las comunicaciones industriales y la interoperabilidad entre aplicaciones cliente, dispositivos industriales y sistemas.

Dispone de una gran variedad de drivers e interfaces (ver Figura 2-9) que permiten conectar con los dispositivos de campo y sistemas superiores.

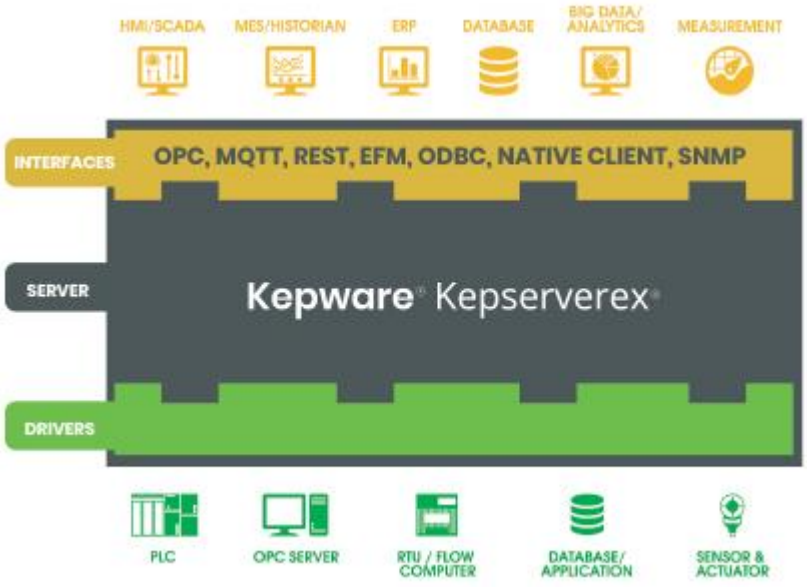


Figura 2-9. Diagrama que muestra los drivers y las interfaces de KEPServerEX. Fuente: kepware.com

KEPServerEX dispone de una interfaz gráfica para interactuar y configurar el servidor (ver Figura 2-10). Pero también dispone de una API de configuración, que permite configurar el servidor desde un cliente REST conectado de forma remota.

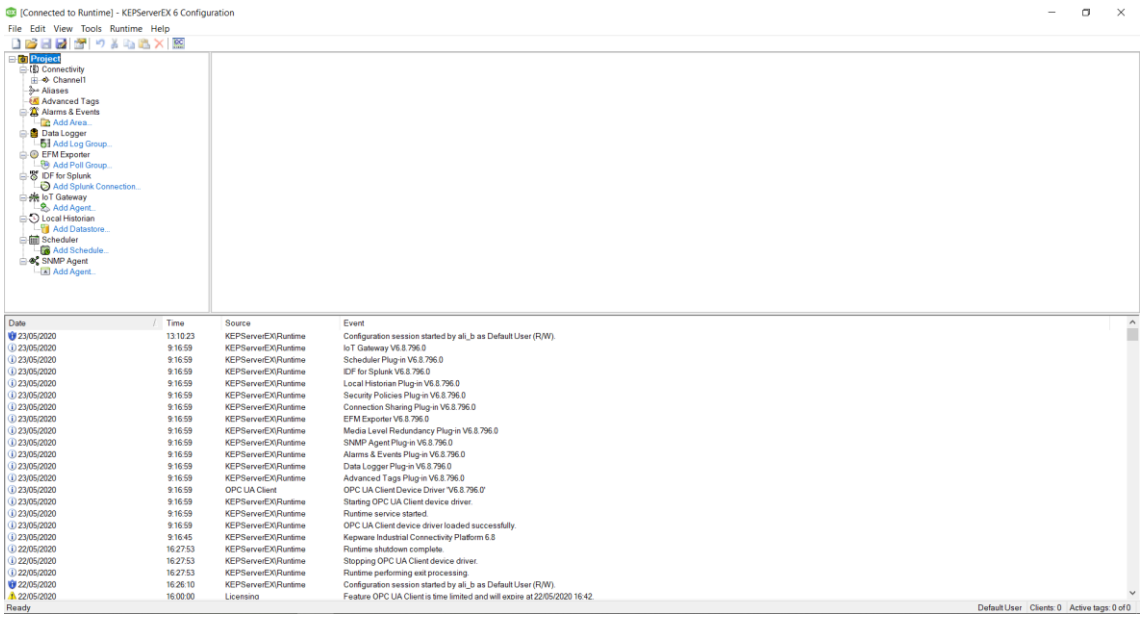


Figura 2-10. Interfaz gráfica de configuración del KEPServerEX.

2.4. Thingworx Kepware Edge

En el mercado existen varios productos para centralizar la información, pero la mayoría son para máquinas con sistema operativo Windows y para máquinas que requieren gran capacidad de computación.

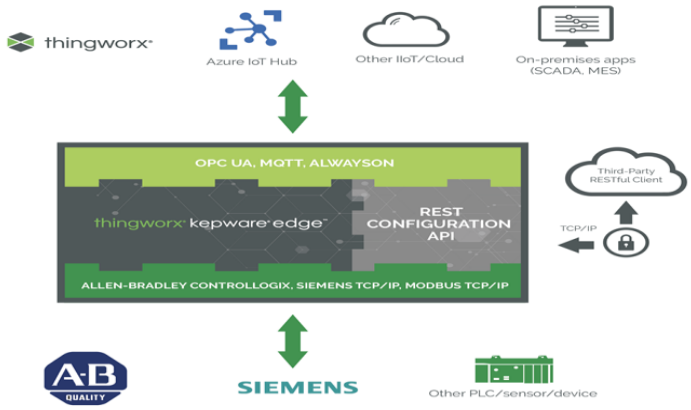


Figura 2-11. Diagrama que muestra los drivers y las interfaces de Thingworx Kepware Edge. Fuente: kepware.com

ThingWorx Kepware Edge es un servidor OPC para sistema operativos Linux. Ha sido diseñado por Kepware, que es el mismo fabricante que desarrolló KEPServerEX. Con lo cual dispone de algunas de las mismas características que el KEPServerEX. Permite la conectividad directamente con la máquina, dispositivo o sensor. Utiliza protocolos seguros, eficientes y preparados para IoT como son el OPC UA o MQTT, para conectarse a plataformas o aplicaciones locales, remotas y basadas en la nube (ver Figura 2-11).

Para configurar este servidor solamente existe la API de configuración. Y no dispone de una interfaz gráfica como la versión del servidor OPC para Windows.

A continuación, se explicarán unos conceptos básicos, para una mejor comprensión de las funcionalidades y para la familiarización con **la terminología específica** de este servidor OPC.

2.5. Conceptos básicos

Todos los conceptos básicos que se explicarán a continuación han sido consultados en un documento técnico (ver Anexo I) creado por el fabricante Kepware. En este documento se detalla toda la terminología específica de ThingWorx Kepware Edge. Mediante los siguientes conceptos básicos se explicarán algunas de las propiedades de los objetos de ThingWorx Kepware Edge, pero para una mejor comprensión se recomienda consultar el documento técnico .

Driver

Un driver **define el protocolo de comunicación y la interfaz física**. Servidor OPC se comunica con las fuentes de datos utilizando un driver que permite la comunicación por una interfaz física y el protocolo correspondiente.

Ejemplos de fuentes de datos:

- Remote Terminal Units (RTUs)
- Base de datos

- Programmable Logic Controllers (PLCs)
- Computer Numerical Control (CNC) Machines

Protocolo de comunicación: es un sistema de reglas que permite que dos o más dispositivos o sistemas se comuniquen entre ellas para transmitir información.

Interfaz física: es el medio de conexión entre el servidor y el hardware o fuente de datos.

Figura 2-12. Cable Ethernet conectado a un switch.

Ejemplos de medio de conexión:

- Ethernet
- Puerto Serial
- Modems (comunicación inalámbrica)
- Application Programming Interface (API)

En definitiva**, un driver tiene dos componentes principales,** que son el **protocolo** de comunicación y la **interfaz física**, que permiten diferenciar un driver del otro.

Ejemplos de drivers:

- Modbus TCP/IP Ethernet
- Mitsubishi RTU Serial
- Allen-Bradley ControlLogix Ethernet
- Omron NJ Ethernet

Interfaz

Una vez que los datos se recogen desde el dispositivo mediante el driver correspondiente, estos se sirven a través de la **interfaz** de cliente necesaria.

Ejemplos de interfaces:

- OLE for Process Control (OPC)
- Dynamic Data Exchange (DDE)
- Machine-To-Machine “M2M” "Internet of Things" connectivity protocol (MQTT)

Canal

Un canal utiliza un driver para definir una “ruta” para llegar hasta la fuente de datos. Representa un medio de comunicación desde el ordenador a uno o más dispositivos externos. Un canal se utiliza para representar un puerto serie, una tarjeta de red instalada en el ordenador o un socket Ethernet.

Dependiendo del tipo de canal este tiene varios parámetros “propiedades”, algunos comunes para todos los canales, pero también existe algunos parámetros específicos para cada tipo de canal. Algunas de las propiedades más relevantes son (ver Tabla 2-1):

Tabla 2-1. Algunas propiedades de un canal.

Propiedad	Descripción
Name	Nombre específico para identificar el canal
Description	Una breve descripción o comentario.
Driver	Indica el tipo de driver al cual pertenece este canal.
Diagnostics Capture	Sirve para guardar o no la información para diagnosticar un canal.

Dispositivo

Si el canal es una ruta, el dispositivo es el destino que representa a una fuente de datos. Los dispositivos representan los PLC, Base de datos u otro hardware con el que se comunica el servidor.

Dependiendo del tipo de dispositivo, este tiene varios parámetros “propiedades”. Algunos comunes para todos los canales, pero también existe algunos parámetros específicos de cada tipo de dispositivo. Algunas de las propiedades más relevantes son (ver Tabla 2-2):

Tabla 2-2. Algunas propiedades de un dispositivo.

Propiedad	Descripción
Name	Nombre específico para identificar el dispositivo.
Description	Una breve descripción o comentario.
Driver	Indica el tipo de driver al cual pertenece el canal de este dispositivo.
Channel Assignment	El nombre del canal el cual pertenece este dispositivo.
ID	La dirección o el nodo donde se encuentra este dispositivo.
Simulated	Una funcionalidad específica para algunos drivers, que permite simular un dispositivo.
Scan Mode	Especifica el método de escaneo de los tags de este dispositivo.

Connect Timeout	Define la máxima cantidad de tiempo en segundos que se permite al dispositivo para establecer una conexión.
Request Timeout	Tiempo en milisegundos que espera el driver para que el dispositivo pueda responder.
Attempts Before Timeout	Numero de intentos que driver realiza para conectar con el dispositivo antes de considerar que el dispositivo se encuentra en un estado de error.
Auto-Demotion	Permite definir un número determinado de intentos, que el driver realizará para comunicar con el dispositivo. Después de estos intentos fallidos el driver esperará un tiempo especificado en milisegundos para volver a intentar comunicar con el dispositivo.

Tag

Un Tag representa la dirección o referencia de una variable dentro de un dispositivo. Cada tag tiene varios parámetros o propiedades asignadas, que sirven para definir las características de este (ver Tabla 2-3).

Tabla 2-3. Las propiedades de un tag.

Propiedad	Descripción
Name	Nombre específico para identificar este tag.
Description	Una breve descripción o comentario.
Address	La referencia o la ubicación del registro a leer o escribir.
Data Type	El tipo de dato: • BCD • Boolean • Byte • Char • Double • DWord • Float • LBCD • LLong • Long • QWord • Short • String • Word • Otros
Client Access	Indica el tipo de acceso que tiene el servidor para acceder a este dato. Por ejemplo, el tag es de escritura o lectura.
Scaling	Es un conjunto de propiedades que permiten escalar el valor de este tag.

Grupo de Tags

Permite organizar etiquetas similares en grupos definidos por el usuario. También tiene sus propiedades correspondientes, pero las más relevantes son el nombre y una breve descripción.

Event Logger

Es una lista donde se registran todos los eventos producidos por el servidor OPC. Cada evento registrado por el servicio Event Log del servidor tiene 4 propiedades (ver Tabla 2-4).

Tabla 2-4. Propiedades de un evento registrado por el servidor OPC.

Propiedad	Descripción
Timestamp	Indica el tiempo de cuando fue registrado este evento
Event	Esta propiedad indica que tipo de evento ha producido. Pueden ser de 4 tipos, que son Warning, Security, Information y Error.
Source	Desde que fuente proviene este evento.
Message	El mensaje descriptivo que explica brevemente el motivo del evento.

Proyecto

Un proyecto contiene toda la configuración necesaria para que el servidor pueda interactuar con los dispositivos de campo y las interfaces.

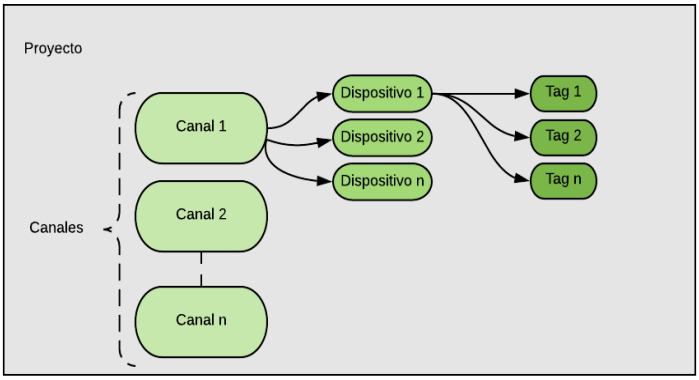


Figura 2-13. La jerarquía de los objetos dentro de un proyecto.

El proyecto está compuesto por una serie de "carpetas" en una determinada jerarquía (ver Figura 2-13). Por ejemplo, en la carpeta principal del proyecto hay una carpeta denominada "canales", que es la que contiene todos los canales creados en forma de subcarpetas. Dentro de cada canal están todos los dispositivos creados para ese canal. Y finalmente dentro de cada dispositivo se encuentran los tags de cada dispositivo.

El funcionamiento de las dos versiones de este servidor OPC es idéntico. Pero la versión de Linux al ser nueva aún no tiene implementadas tantas prestaciones como su hermano mayor el

KEPServerEX. A continuación, se realizará una comparativa entre las dos versiones para ver las ventajas y desventajas de cada servidor OPC.

2.6. KEPServerEX vs ThingWorx Edge

En la siguiente tabla (ver **Tabla 2-5**) se detallan las diferencias y especificaciones de cada versión del servidor OPC.

Tabla 2-5. Comparativa entre KEPServerEX y ThingWorx Kepware Edge.

KEPServerEX	ThingWorx Kepware Edge
Se instala en sistemas operativos de Microsoft, que requieren una licencia.	Diseñado para sistemas operativos basado en Linux (software libre).
Se puede instalar en ordenadores y máquinas virtuales.	Se puede instalar en un Single Board Computer (SBC). Ordenador y/o servidor que no requiere gran capacidad de computación.
Mas de 150 drivers. • ModBus TCP/IP Ethernet • Simulator • Siemens TCP/IP Ethernet • Allen-Bradley ControlLogix • DDE Client • OPC UA Client • OPC DA Client • Otros	Actualmente solo dispone de 4 drivers. • ModBus TCP/IP Ethernet • Simulator • Siemens TCP/IP Ethernet • Allen-Bradley ControlLogix
Fácil de configurar, a través de una aplicación de configuración. También dispone de API de configuración.	Solamente se puede configurar mediante una REST API de configuración.
Mas de 8 interfaces	Dispone de las interfaces OPC UA, MQTT y AlwaysON.

La aplicación de configuración que se va a desarrollar en este proyecto utilizará la API REST de configuración de los dos servidores para configurarlos. Para comprender mejor cómo funciona la API REST específica de este servidor OPC, a continuación, ¿se hablará sobre que son las API REST?, ¿cómo funcionan? y ¿para qué sirven?

2.7. ¿Qué es una REST API?

2.7.1. Conceptos básicos

Application Programming Interface (API)

Es un conjunto de funciones y especificaciones definidas por el desarrollador de una aplicación para que esta pueda comunicar, interactuar e intercambiar información con otra aplicación. De manera que hace de interfaz entre programas, tal y como una interfaz de usuario facilita la interacción de un humano con un software.

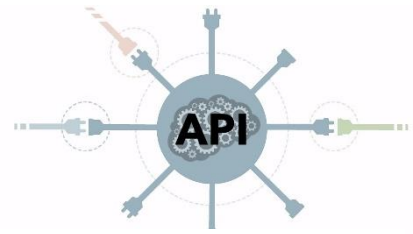


Figura 2-14. API REST. Fuente: tecnologias-informacion.com

REpresentational State Transfer (REST)

Es una lógica de restricciones y recomendaciones para construir una API. Dicho de otro modo, es un estilo de arquitectura.

2.7.2. REST API

REST API es una API creada siguiendo las restricciones y lógica de REST. Funciona estrictamente bajo la arquitectura cliente – servidor y la comunicación se realiza utilizando HTTP como protocolo de comunicación.

Para entender mejor lo que es una API REST se apoyará de un ejemplo basado en una actividad cotidiana.

Supongamos que un cliente entra en un restaurante para cenar. El cliente llama al camarero para pedir la cena y el camarero enseña la carta con todas las especialidades de la casa. El cliente elige un plato y se queda a la espera que el camarero traiga la comida. Una vez el cocinero ha acabado de preparar la comida, llama al camarero para que se la lleve al cliente.

Una API REST funciona de la misma manera, donde un cliente intercambia información con un servidor “cocinero” con la ayuda de un API REST “camarero”. Lo único que tiene este cliente es una carta con todas las posibles peticiones que puede hacer al servidor. Pero no tiene por qué saber cómo el servidor procesa esta información y devuelve una respuesta.

El servidor software puede estar programado en un lenguaje de programación diferente al del cliente. Pero aun así es posible el intercambio de información gracias a la API REST. Tanto el cliente como servidor se comunican entre ellos utilizando un protocolo de comunicación común (HTTP). Sobre este protocolo de comunicación se construye la API REST, con un conjunto de reglas y recomendaciones para desarrollar las posibles peticiones.

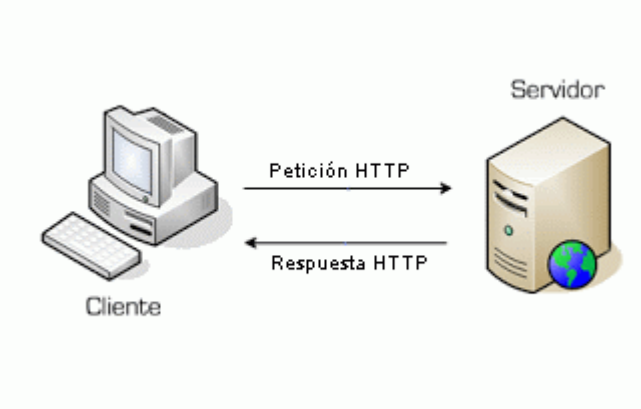


Figura 2-15. Arquitectura cliente servidor - comunicación por HTTP.

Un cliente lanza una petición al servidor (ver Figura 2-15) ya sea para intercambiar información o ejecutar una función ya definida en el servidor. El servidor procesa la petición y acorde a esta devuelve una respuesta al cliente.

Para que el cliente pueda realizar una petición necesita un verbo HTTP, una URI única y los datos, ya sean en formato XML, JSON o Binario.

Verbo HTTP: el verbo define el tipo de petición que se va a realizar al servidor con especificaciones REST. Existen varios tipos de peticiones "operaciones" HTTP, pero los más utilizados son los siguientes:

1. **POST:** para crear nuevos recursos.
2. **GET:** para consultar un conjunto (listado) o un único recurso en concreto.
3. **PUT:** sirve para modificar
4. **DELETE:** para borrar.

URI: es la dirección única de un determinado recurso.

Body: es la información necesaria para que el servidor pueda satisfacer una petición. Esta información también incluye los datos necesarios para la autentificación del cliente.

El fabricante Kepware ha elaborado un documento técnico (ver Anexo I), que explica detalladamente el funcionamiento de la API REST. Este es el documento principal que se ha utilizado para programar la aplicación de configuración.

Para comprender mejor el funcionamiento de esta arquitectura a continuación se explicará detalladamente un ejemplo de petición lanzada por un cliente.

2.7.3. Ejemplo: ¿Cómo crear un nuevo canal en el servidor OPC?

Primeramente, debe definirse el tipo de petición que se va a realizar al servidor. En este caso para crear un canal nuevo la petición será de tipo POST.

A continuación, la dirección o la ruta donde se va a crear este canal:

```
https://<hostname_or_ip>:<port>/config/v1/project/channels
```

Finalmente, toda la información necesaria en formato JSON que requiere el servidor para definir un canal:

```
{
"common.ALLTYPES_NAME": "MyChannel",
"servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Modbus TCP/IP Ethernet"
}
```

3. Instalación de ThingWorx Kepware Edge

3.1. Requisitos del sistema para ThingWorx Kepware Edge

El ThingWorx Kepware Edge actualmente solo funciona en plataformas con arquitectura de X86_64 y sistemas operativos basados en Linux. El fabricante recomienda utilizar Ubuntu 18.04 LTS o una versión superior.

Ubuntu es un sistema operativo de código abierto. Es una distribución de Linux basada en Debian. Se puede instalar en ordenadores de escritorio y servidores.

Prerrequisitos:

- Computadora con procesador de 64bit (arquitectura X86_64).
- Tener instalado sistema operativo Ubuntu de 64bits.

  En este proyecto se ha utilizado un ordenador portátil (ver Figura 3-1) de 64 bits, con la versión 18.04.4 LTS de Ubuntu.

Figura 3-1. Captura de pantalla que muestra las características de la máquina.

- Verificar que el sistema operativo está actualizado.

  Para comprobar si el sistema operativo requiere una actualización e instalar las ultimas actualizaciones se debe ejecutar el siguiente comando desde la terminal.

```
$ sudo apt update
```

- Tener instalado los últimos paquetes de Linux Standard Base (LSB).

  Para instalar los componentes Linux Standard Base en Ubuntu, se puede ejecutar el siguiente comando desde la terminal.

```
$ sudo apt install lsb
```

- En caso de utilizar la interfaz MQTT hay que instalar Java Runtime Environment.

```
$ sudo apt install default-jdk
```

- Tener el instalador de ThingWorx Kepware Edge.

  Se puede descargar desde la página web oficial del fabricante, en este se ha descargado la última versión "thingworx_kepware_edge_1.1.796.0.run".

3.2. Instalación

La instalación se debe llevar a cabo por un usuario con **los permisos *root***. El instalador dispone de una interfaz gráfica para facilitar el proceso de instalación. Pero a su vez es compatible con líneas de comandos.

1. Antes de proceder a ejecutar el instalador, se ha de verificar que este tiene los permisos (ver Figura 3-2) para ser ejecutado como un programa.

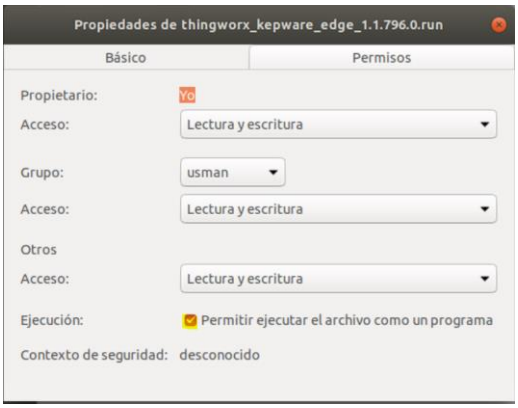


Figura 3-2. El check que hay que marcar para dar los permisos.

2. Desde la carpeta que contiene el archivo ".run", se debe hacer click derecho y abrir un nuevo terminal.

3. En el terminal se debe ejecutar el siguiente comando:

```
$ sudo ./ thingworx_kepware_edge_1.1.796.0.run
```

   Este comando lanzará el asistente de instalación (ver Figura 3-3) de ThingWorx Kepware Edge.

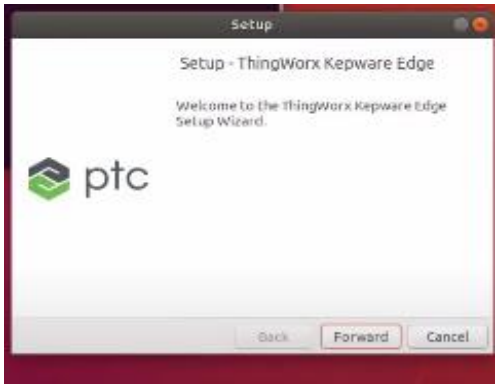


Figura 3-3. Asistente de instalación de ThingWorx Kepware Edge.

4. Durante el proceso de instalación el asistente pedirá una contraseña (ver Figura 3-4) para la cuenta de Administrator:

Figura 3-4. Contraseña para la cuenta Administrator.

Es muy importante recordar esta contraseña, ya que todos los cambios en el servidor requieren de los permisos de Administrator.

5. Es recomendable dejar todos los valores por defecto, hasta llegar a finalizar la instalación (ver Figura 3-5).

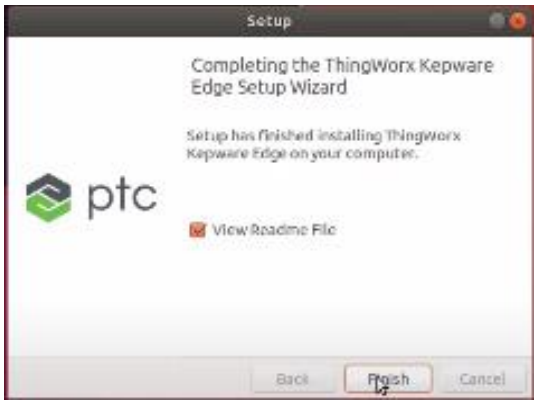


Figura 3-5. Pantalla final de la instalación.

3.3. Servicios

Para verificar que la instalación se ha realizado correctamente y que el servidor ya está en marcha se puede consultar el estado de los servicios (ver Figura 3-6) bajo los cuales corre el servidor.

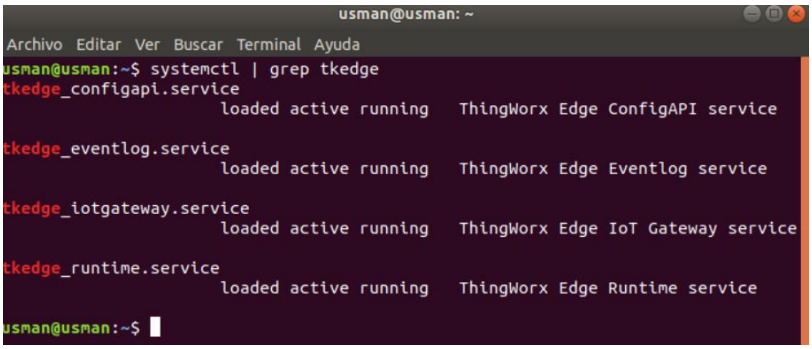


Figura 3-6. Captura de la pantalla que muestra los 4 servicios en terminal.

Desde un terminal ejecutar el siguiente comando:

```
systemctl |grep tkedge
```

El servidor tiene 4 servicios asociados:

Runtime: es el proceso principal del servidor, es decir, el motor que corre todo el servidor. Este servicio contiene el proyecto actual, se encarga de comunicar y obtener los datos desde los dispositivos de campo y provee estos datos a las interfaces.

Configuration API: provee una interfaz REST web para interactuar y configurar el servidor.

IoT Gateway: permite que los agentes MQTT puedan interactuar con el Runtime y publicar los datos a un bróker MQTT.

Event Log: este servicio es el encargado de registrar y administrar todos los eventos y mensajes producidos por los otros servicios.

3.4. Licencias en ThingWorx Kepware Edge

ThingWorx Kepware Edge en modo demo (sin licencia) es 100 % funcional durante dos horas. Una vez transcurrido ese periodo, dejará de funcionar. Una forma de seguir trabajando en modo demo es, reiniciar el servicio Runtime del servidor OPC para obtener otras dos horas más de funcionamiento.

Las licencias de ThingWorx Kepware Edge están basadas en el número de tags a utilizar en el servidor. Esta funcionalidad es proporcionada por una llave hardware USB, que tiene asociado un archivo de certificado que determina el número de tags que están permitidos por un driver.

Tipos de licencias limitadas por el número de tags:

- 100 tags

- 750 tags
- 1500 tags
- Ilimitado

3.4.1. Instalar una licencia

a. Se debe de copiar el archivo .lic correspondiente a la llave hardware (USB) en la siguiente ruta:

<ruta de instalación de ThingWorx Edge> ./config/License.

b. Insertar la llave hardware en la máquina y observar si se enciende la luz roja en la llave.

c. Reiniciar el servicio Runtime, enviando el siguiente comando desde la terminal:

sudo systemctl restart tkedge_runtime.service

4. Pruebas de configuración del servidor OPC vía API REST

4.1. Arquitectura

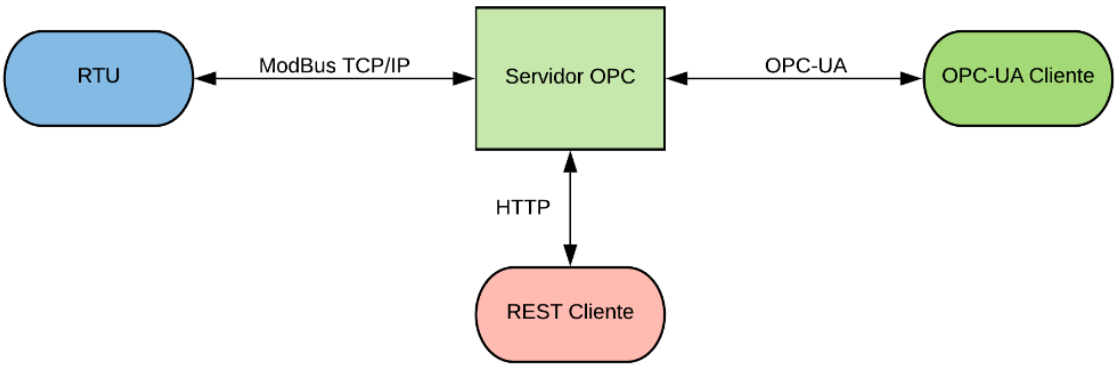


Figura 4-1.Arquitectura

Para comprender mejor el funcionamiento de este servidor, se ha montado una arquitectura (ver **Figura 4-1**) donde, los datos prevenientes desde una RTU (Remote Terminal Unit) son recogidos por el servidor OPC. Para configurar el servidor OPC se ha utilizado un cliente REST (Postman). Y finalmente los datos recogidos desde la RTU son consultados por un cliente OPC-UA (UAExpert).

Para montar esta arquitectura se han utilizado dos ordenadores portátiles y una RTU (ver **Figura 4-2**). Uno de los ordenadores tiene Linux como sistema operativo y en este se ha instalado el servidor OPC. El segundo ordenador tiene instalado Windows 10 como sistema operativo. En este segundo ordenador se encuentran instalado el Postman y UA Expert. A su vez este ordenador tiene instalado TwinSoft que es un software para programar la RTU.

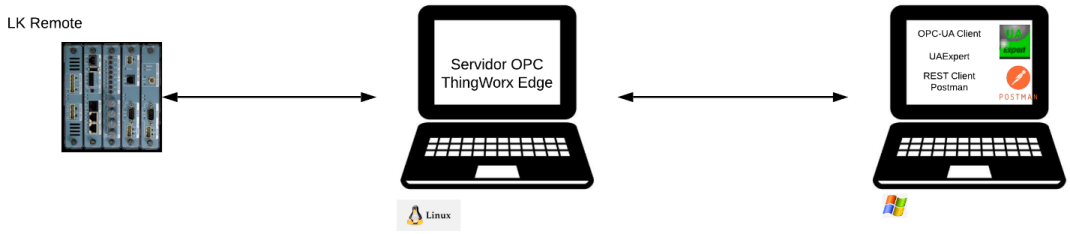


Figura 4-2. El material empleado para montar la arquitectura

Material empleado:

- Ordenador portátil con el servidor OPC.
- La unidad terminal remota (RTU) MS de LKRemote + Módulo de simulación

- Fuente de alimentación de 24 V, para la RTU.
- Cables Ethernet para la comunicación entre la RTU y el ordenador donde estará alojado el servidor OPC.
- Ordenador portátil con sistema operativo Windows 10 con los programas:
    - TwinSoft
    - Postman
    - UAExpert

Durante estas pruebas se han realizado varias peticiones de configuración a la REST API del servidor OPC. Pero para comprender el funcionamiento de la API REST, en esta memoria de forma breve se explicará un único ejemplo. Donde utilizando la arquitectura montada, en el servidor OPC se configurarán dos tags provenientes de la RTU. De manera que el cliente final (UAExpert) podrá consultar estos dos tags. Los dos ordenadores y la RTU están conectados a la misma red, para que se puedan comunicar entre ellos.

4.2. Preparación de la RTU

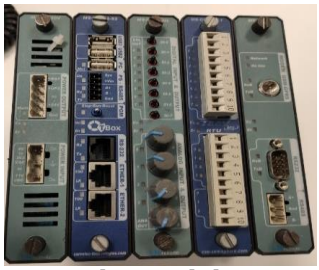


Figura 4-3. RTU

RTU (ver **Figura 4-3**) es una terminal de acceso remoto que de forma nativa se comporta como un servidor ModBus. Tiene conectado un módulo de simulación de entradas y salidas. El módulo simulador de entradas y salidas tiene 8 entradas digitales, 8 salidas digitales, 4 entradas analógicas y 4 salidas analógicas.

TwinSoft es el programa que se ha utilizado para configurar la RTU. Se carga un programa a la RTU, donde están definidos varios tags vinculados a entradas y salidas del módulo simulador.

De manera que en la dirección "registro" 545 del servidor ModBus se puede leer y escribir la salida digital 0, del módulo simulador. Y de la misma manera en el registro 10513 se puede leer la entrada digital 0.

4.3. Configuración del Servidor OPC

La única forma de configurar es servidor OPC, es mediante la API REST. Por lo tanto, se ha utilizado Postman que es un cliente REST para interactuar con el servidor.

Para configurar el servidor primeramente se debe crear y cargar un proyecto al servicio Runtime del servidor OPC. El asistente de instalación de ThingWorx Kepware Edge permite cargar un proyecto "ejemplo" por defecto que se carga y se ejecutará directamente en el Runtime después de la instalación. De esta manera, durante la instalación ya hemos cargado un proyecto a nuestro servidor OPC.

Para la autentificación se debe utilizar el mismo usuario **"Administrator"** y la contraseña que se ha configurado en la instalación.

La API REST de configuración utiliza los puertos 57513 (habilitado por defecto) para HTTPS y **57413** para HTTP (deshabilitado por defecto).

Para realizar una petición al servidor REST desde Postman (ver **Figura 4-4**) hay que elegir el tipo de petición (1), definir la dirección URI que también recibe el nombre de Endpoint (2), la

autentificación (usuario y contraseña) (3) y body (4) (ver **Figura 4-5**), si este tipo de petición lo requiere.

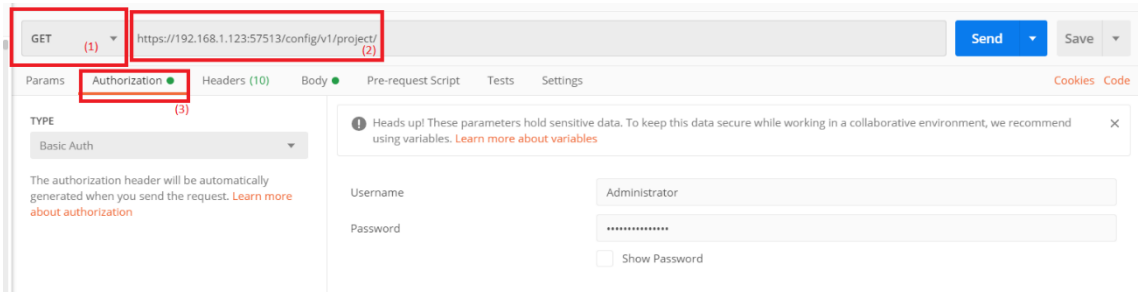


Figura 4-4. Captura de la pantalla, donde se muestra la interfaz gráfica del programa Postman

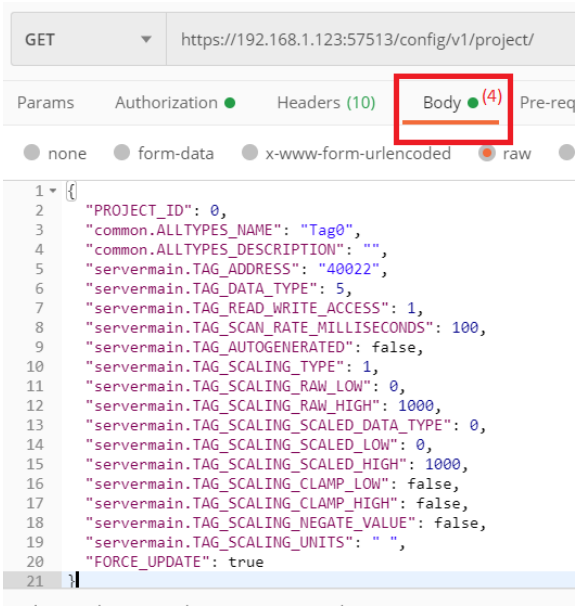


Figura 4-5. Pestaña body.

4.3.1. Configuración de un nuevo driver – Modbus TCP/IP Ethernet

Para esta prueba **se ha creado un canal** utilizando el driver ModBus TCP/IP Ethernet en el servidor OPC. Para crear este canal se lanza la siguiente petición HTTP de tipo POST:

Petición POST:

Endpoint:

```
https://192.168.1.123:57513/config/v1/project/channels
```

Body:

```
{
"common.ALLTYPES_NAME": "RTU_ModBus_Canal",
"servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Modbus TCP/IP Ethernet"
}
```

Un canal dependiendo del tipo de driver, tiene varios parámetros asociados que permiten definir las características de este canal. Para que el servidor pueda procesar esta petición hay que enviar estos parámetros en formato JSON (body).

Para crear un canal nuevo solo es obligatorio incluir el nombre y el tipo de driver en el body de esta petición, los parámetros restantes tomarán un valor por defecto si estos no son incluidos en la petición.

Una vez lanzada la petición, el servidor procesa y devuelve una respuesta también en formato JSON. Si el canal se ha creado correctamente, el body que devuelve el servidor estará en vacío. Pero por ejemplo si alguno o algunos de los parámetros enviados no son los correctos o este nombre de canal ya fue creado previamente, el servidor, en el body de la respuesta incluirá el motivo por el cual, el canal no fue creado.

Para comprobar que el canal se ha creado correctamente se puede lanzar otra petición de tipo GET referente a este canal ("**RTU_ModBus_Canal**").

En este caso el Endpoint debe de incluir el nombre del canal.

Petición GET:

Endpoint:

https://192.168.1.123:57513/config/v1/project/channels/**RTU_ModBus_Canal**

La respuesta del servidor:


```
        {
  "PROJECT_ID": 1973086675,
  "common.ALLTYPES_NAME": "RTU_ModBus_Canal",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Modbus TCP/IP Ethernet",
  "servermain.CHANNEL_DIAGNOSTICS_CAPTURE": false,
  "servermain.CHANNEL_UNIQUE_ID": 3164000198,
  "servermain.CHANNEL_ETHERNET_COMMUNICATIONS_NETWORK_ADAPTER_STRING": "",
  "servermain.CHANNEL_WRITE_OPTIMIZATIONS_METHOD": 2,
  "servermain.CHANNEL_WRITE_OPTIMIZATIONS_DUTY_CYCLE": 5,
  "servermain.CHANNEL_NON_NORMALIZED_FLOATING_POINT_HANDLING": 0,
  "servermain.CHANNEL_COMMUNICATIONS_SERIALIZATION_VIRTUAL_NETWORK": 0,
  "servermain.CHANNEL_COMMUNICATIONS_SERIALIZATION_TRANSACTIONS_PER_CYCLE
": 1,
  "servermain.CHANNEL_COMMUNICATIONS_SERIALIZATION_NETWORK_MODE": 0,
  "modbus_ethernet.CHANNEL_USE_ONE_OR_MORE_SOCKETS_PER_DEVICE": 1,
  "modbus_ethernet.CHANNEL_MAXIMUM_SOCKETS_PER_DEVICE": 1
        }
```


El servidor devuelve todos los parámetros de este canal, lo que significa que el canal fue creado correctamente.

Una vez creado el canal, a continuación, **se crea un dispositivo** que va a representar a la RTU. Esta petición deberá incluir parámetros y características de la RTU, como por ejemplo el nombre, la dirección IP y a qué tipo de driver pertenece este dispositivo.

Petición POST:

Endpoint:

```
https://192.168.1.123:57513/config/v1/project/channels/RTU_ModBus_Canal/devices
```

Body:

```
{
"common.ALLTYPES_NAME": "RTU",
"servermain.DEVICE_ID_STRING": "<192.168.1.99>.0",
"servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Modbus TCP/IP Ethernet"
}
```

La respuesta del servidor en este caso fue, un body en vacío. Con lo cual significa que el dispositivo fue creado correctamente.

Finalmente, se crean los dos tags que se han definido en la preparación de la RTU (ver **Preparación de la RTU**).

Nota: el JSON puede incluir varios objetos "canal, dispositivo, tags o grupos de tags" en una única petición.

Petición POST:

Endpoint:

```
https://192.168.1.123:57513/config/v1/project/channels/RTU_ModBus_Canal/devices/RTU/tags
```

Body:

```
[{
"common.ALLTYPES_NAME": "Tag0",
"servermain.TAG_ADDRESS": "00545"
}
{
"common.ALLTYPES_NAME": "Tag1",
"servermain.TAG_ADDRESS": "10513"
}]
```

4.3.2. Configuración de una nueva interfaz – OPC UA

Para que el servidor OPC pueda servir los datos a un cliente final, se ha de configurar una interfaz. ThingWorx Kepware Edge dispone de varias interfaces, pero en esta prueba se configurará un servidor OPC-UA.

La interfaz OPC-UA está habilitada de forma nativa, pero se puede deshabilitar desde la configuración del proyecto que está cargado en el Runtime.

Realizando una consulta de tipo GET al siguiente Endpoint se pueden consultar todos los OPC-UA Endpoints configurados en el servidor OPC.

Petición GET:

Endpoint:

```
https://192.168.1.123:57513/config/v1/admin/ua_endpoints
```

Respuesta del servidor:


```
{
   "common.ALLTYPES_NAME": "local",
   "common.ALLTYPES_DESCRIPTION": "Available adapters: Default; wlp2s0:192.168.1.123; enp1s0:<
disconnected>; localhost",
   "libadminsettings.UACONFIGMANAGER_ENDPOINT_ENABLE": true,
   "libadminsettings.UACONFIGMANAGER_ENDPOINT_ADAPTER": "localhost",
   "libadminsettings.UACONFIGMANAGER_ENDPOINT_PORT": 49330,
   "libadminsettings.UACONFIGMANAGER_ENDPOINT_URL": "opc.tcp://127.0.0.1:49330",
   "libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_NONE": true,
   "libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC128_RSA15": 0,
   "libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC256": 0,
   "libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC256_SHA256": 0
}
```


Como se puede observar el servidor OPC, nos ha devuelto un OPC UA Endpoint llamado local, con todos los otros parámetros correspondientes a este OPC UA Endpoint. Este Endpoint tiene configurado localhost como adaptador de red. Con lo cual es solo accesible por clientes OPC UA localizado en la misma máquina.

Para que el servidor sea accesible desde fuera de nuestra máquina local, por ejemplo, desde el segundo ordenador utilizado para esta prueba, tenemos que crear un OPC UA Endpoint con un adaptador de red que tenga acceso a nuestra red.

En la petición anterior el parámetro Description, contiene todos los adaptadores presentes en nuestra máquina Linux. El adaptador de red que tiene acceso a nuestra red es el wpl2s0, con la IP 192.168.1.123 asignada.

Para crear el nuevo Endpoint en el servidor OPC UA, se ha realizado la siguiente petición de tipo POST.

Petición POST:

Endpoint:

```
https://192.168.1.123:57513/config/v1/admin/ua_endpoints
```

Body:

```
{
        "common.ALLTYPES_NAME": "serverOPC-UA",
        "libadminsettings.UACONFIGMANAGER_ENDPOINT_ADAPTER": "wlp2s0"
}
```

Los únicos parámetros obligatorios para crear un OPC UA Endpoint, son el nombre y el adaptador de red. Todos los parámetros restantes tomarán un valor por defecto, en caso de que no se hayan incluido en el body de la petición.

Finalmente, para verificar que el OPC UA Endpoint se ha creado correctamente, se realiza una petición de tipo GET.

Petición GET:

Endpoint:

```
https://192.168.1.123:57513/config/v1/admin/ua_endpoints
```

Respuesta del servidor:


```
[
        {
                "common.ALLTYPES_NAME": "local",
                "common.ALLTYPES_DESCRIPTION": "Available adapters: Default;
wlp2s0:192.168.1.123; enp1s0:<disconnected>; localhost",
                "libadminsettings.UACONFIGMANAGER_ENDPOINT_ENABLE": true,
                "libadminsettings.UACONFIGMANAGER_ENDPOINT_ADAPTER": "localhost",
                "libadminsettings.UACONFIGMANAGER_ENDPOINT_PORT": 49330,
                "libadminsettings.UACONFIGMANAGER_ENDPOINT_URL":
"opc.tcp://127.0.0.1:49330",
                "libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_NONE": true,

"libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC128_RSA15": 0,
                "libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC256": 0,

        "libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC256_SHA256": 0
        },
        {
                "common.ALLTYPES_NAME": "serverOPC-UA",
                "common.ALLTYPES_DESCRIPTION": "Available adapters: Default;
wlp2s0:192.168.1.123; enp1s0:<disconnected>; localhost",
                "libadminsettings.UACONFIGMANAGER_ENDPOINT_ENABLE": true,
                "libadminsettings.UACONFIGMANAGER_ENDPOINT_ADAPTER": "wlp2s0",
                "libadminsettings.UACONFIGMANAGER_ENDPOINT_PORT": 49330,
                "libadminsettings.UACONFIGMANAGER_ENDPOINT_URL":
"opc.tcp://192.168.1.123:49330",
                "libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_NONE": false,

        "libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC128_RSA15": 0,
                "libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC256": 0,

        "libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC256_SHA256": 2
        }
]
```


Dado que la petición fue realizada preguntando por todos los OPC UA Endpoints disponibles en el servidor OPC UA, el servidor devuelve los dos Endpoints, el llamado local que ya estaba creado previamente y el nuevo que se llama serverOPC-UA, que se ha creado durante este ejemplo.

4.4. Verificación de la configuración del servidor OPC

Una vez configurado el servidor, se ha procedido a verificar esta configuración. Una forma de verificar esta configuración y el correcto funcionamiento del Runtime, es utilizar un cliente OPC-

UA que es el encargado de consultar los dos tags referenciados a una entrada y salida digital de la RUT.

Como cliente OPC-UA, se ha utilizado el UAExpert que se puede descargar desde su página web oficial de forma gratuita.

En UAExpert se agrega una nueva conexión (ver **Figura 4-6**), donde se indica el nombre de la conexión, el OPC-UA Endpoint y la autentificación.

Nota: Por defecto el usuario y la contraseña para autenticar son los mismos que han sido configurado, en la instalación del servidor OPC.

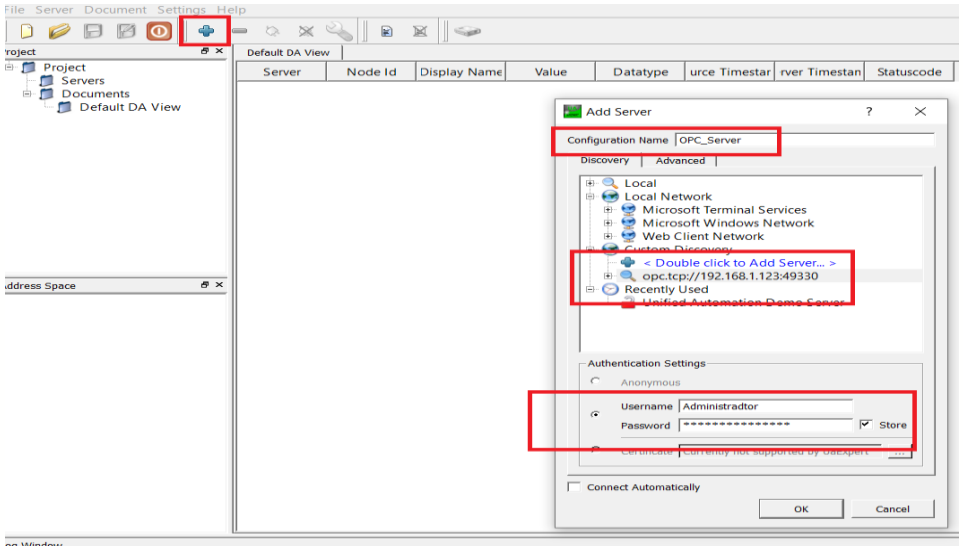


Figura 4-6. Captura que muestra cómo se agrega una nueva conexión en UAExpert.

Una vez conectado con el servidor OPC, UAExpert muestra toda la información disponible (ver **Figura 4-7**) en el servidor en forma de árbol.

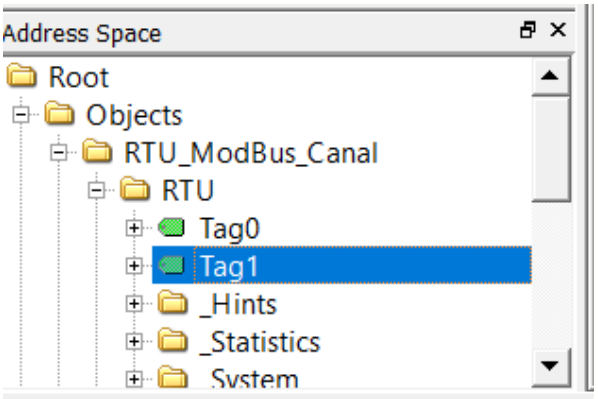


Figura 4-7. Toda la configuración que contiene el proyecto actual, que está cargado en el Runtime.

La carpeta Objects contiene todos los canales creados. Dentro de nuestro canal creado está el dispositivo que contiene los dos Tags.
Una vez localizados los tags, se arrastran hasta la ventana de Default DA View (ver **Figura 4-8**). En esta ventana se puede observar:

Node Id: la ruta donde se encuentra este Tag dentro del servidor OPC.

Display Name: el nombre del tag.
Valor: es el último valor que tiene el servidor OPC de este tag.
Datatype: tipo de dato.
Tiemstamp: el tiempo que indica cuando fue el último cambio de valor de este tag.
Statuscode: indica la calidad, es decir el estado del tag.

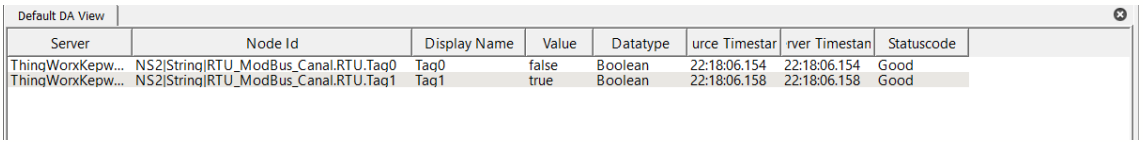


Figura 4-8. Valores recibidos en UAExpert.

4.5. Conclusiones

El usuario final, para cada petición debe de crear una estructura de texto en formato JSON. Esto puede llegar a ser muy tedioso si se tiene que crear una gran cantidad de canales, dispositivos o tags. A su vez cualquier error de sintaxis en la estructura JSON, puede provocar la cancelación de una petición.

La respuesta del servidor también está en forma JSON y devuelve todos los parámetros del objeto consultado. Lo cual dificulta la lectura de la información de más interés.

Por lo tanto, la REST API de configuración del servidor OPC, no es una forma práctica de configurar el servidor para el usuario mediante un cliente REST enviando peticiones en formato JSON de forma “manual”.

La estructura y el funcionamiento en si del servidor OPC es fácil de entender y una vez configurado funciona correctamente.

En base a estas conclusiones, es indudable que existe una necesidad de mejorar este método de configurar el servidor OPC por un usuario vía API REST. Para ello, se debe de diseñar una interfaz gráfica, que sea más fácil y práctica, y que mejore la interacción de usuario con el servidor OPC.

5. Diseño de una aplicación para la gestión de un servidor OPC

5.1. ¿Por qué?

Es evidente que tras observar el funcionamiento de la API REST, la configuración del servidor no es una tarea sencilla y práctica ya que la API de configuración del servidor no permite al usuario interactuar con la misma de una forma amigable. Por ello en este proyecto se va a desarrollar una aplicación de configuración para el servidor OPC.

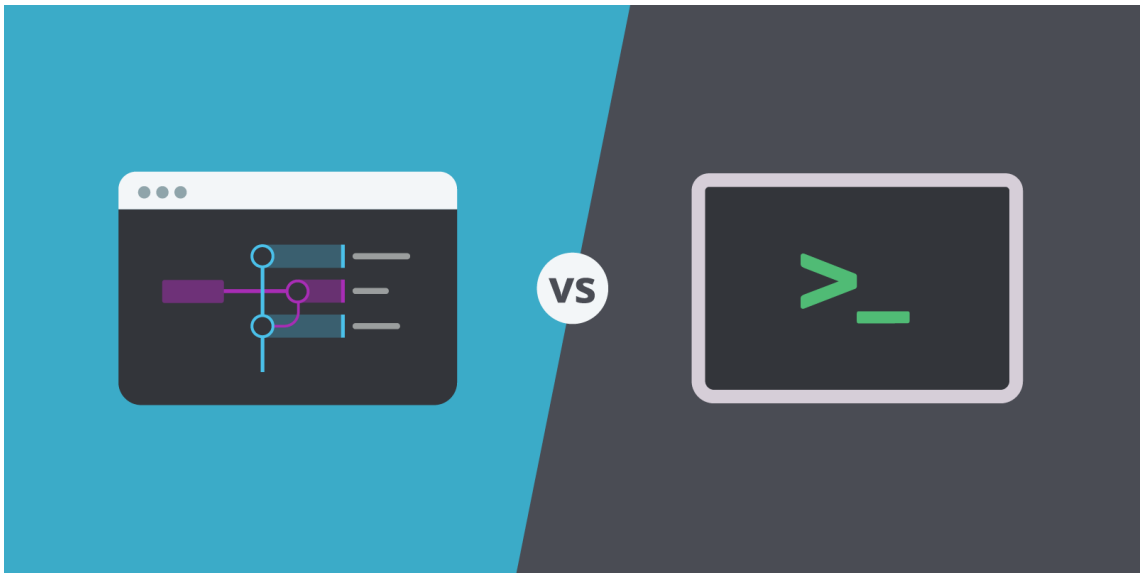


Figura 5-1. Concepto de una GUI. Fuente: muycomputer.com

Esta aplicación tendrá una interfaz gráfica (GUI) (ver **Figura 5-1**) que va a permitir al usuario final, configurar el servidor OPC de una forma más simple y ágil. Por ejemplo, si el usuario quiere crear un canal, solo tendrá que introducir los datos o los parámetros relevantes en la interfaz gráfica, y esta será la encargada de convertir estos parámetros en una estructura de texto JSON correspondiente y enviarla al servidor en forma de petición HTTP. Una vez el servidor OPC procesa esa información y devuelve una respuesta, que también será en formato JSON, la aplicación de configuración también se encargará de convertir esa estructura JSON en información útil y mostrarla al usuario de una forma más visual y comprensible.

La aplicación se comunicará con la API de configuración del servidor OPC utilizando HTTP como protocolo de comunicación.

5.2. Entorno de programación

5.2.1. Entorno de desarrollo integrado (IDE)

En este caso se ha decidido utilizar Visual Studio como entorno de programación, ya que Microsoft Visual Studio es un entorno de desarrollo integrado (IDE) compatible con una gran variedad de lenguajes de programación como, Visual Basic, C++, .NET, C#, Java, Python, PHP, etc.

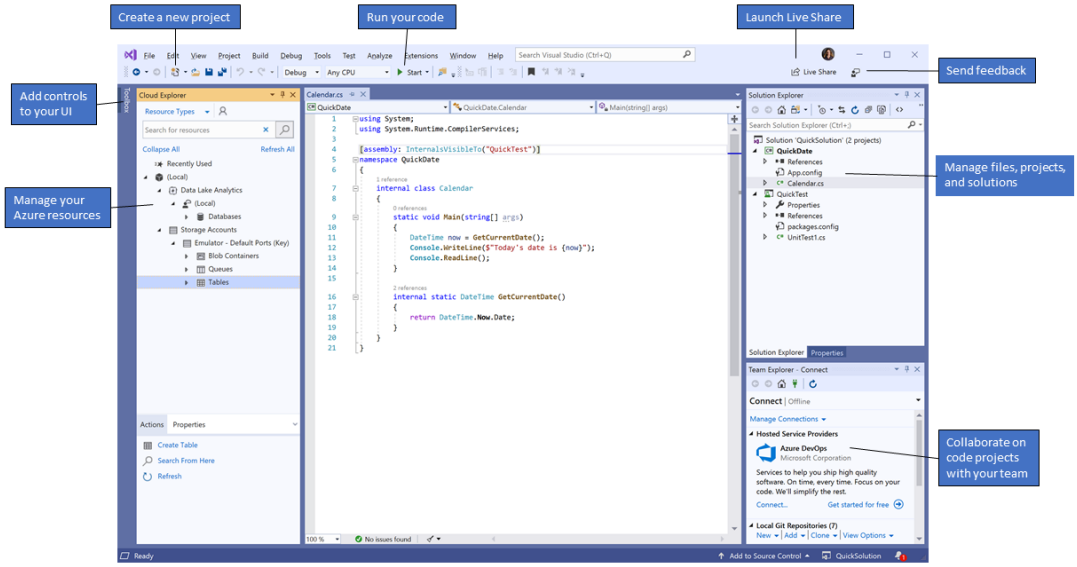


Figura 5-2. Entorno de Microsoft Visual Studio. Fuente: docs.microsoft.com

Visual Studio (ver **Figura 5-2**) es una herramienta desarrollada por Microsoft para plataformas compatibles con .NET. Dispone de una versión comunitaria gratuita con funcionalidades básicas separadas por lenguajes de programación o plataforma. Pero a su vez existe una versión comercial con ciertas características y funcionalidades más avanzadas,

Es un pack con todas las funcionalidades y características necesarias para el desarrollo de un proyecto. Por ejemplo, permite compilar y depurar, ayudas para autocompletar el código, integración con las bases de datos, rastrear errores, dispone de herramientas de implementación, etc.

Una de las ventajas de utilizar Visual Studio es su capacidad de facilitar al usuario, la tarea de diseñar una interfaz gráfica. El "Visual" de Visual Studio es el sinónimo de diseño de la interfaz de usuario visual (arrastrar y soltar). Visual Studio ya de forma nativa incluye una gran variedad de componentes y herramientas, como tales, botones, cuadros de textos, tablas, listas de textos, desplegables, etc. Que son configurables de forma rápida y sencilla sin necesidad de escribir código.

Por ejemplo, si se desea crear un botón (ver **Figura 5-3**) en nuestra aplicación, el programador tan solo tiene que ir a cuadro de herramientas, escoger un botón y arrastrarlo hasta la aplicación a desarrollar.

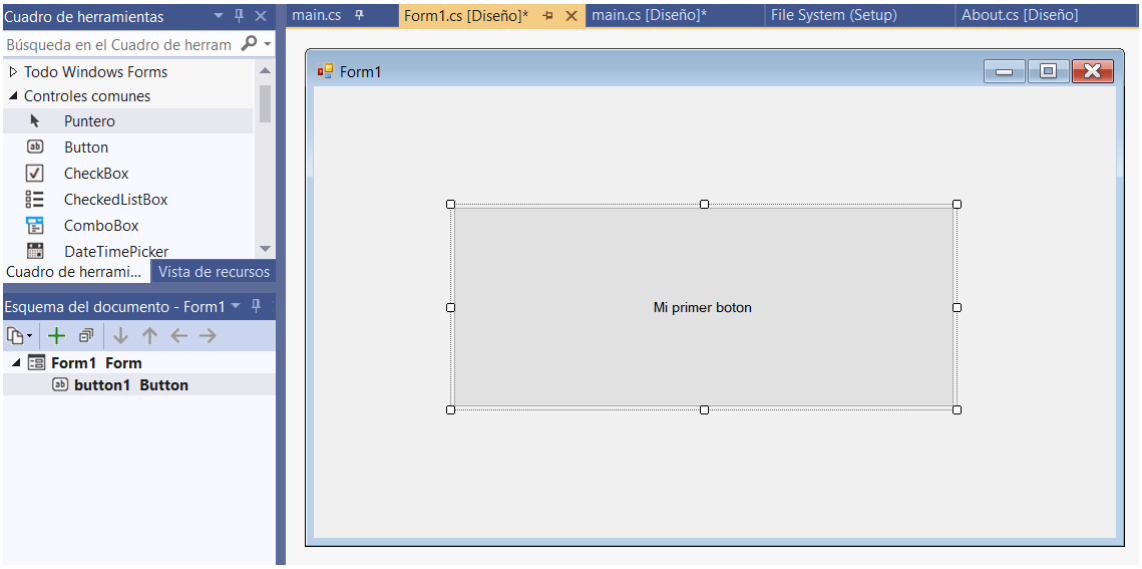


Figura 5-3. Creando el primer botón en Visual Studio

5.2.2. Lenguaje de programación.

Hoy en día hay muchos lenguajes de programación orientados a objetos que podrían llevar a cabo esta tarea. Cada uno con sus ventajas y desventajas, pero esta decisión ya es una cuestión de conocimientos, recursos disponibles y la forma de trabajar de cada programador.

Se ha apostado por C# como lenguaje de programación. Ya que en el departamento de I+D de Logitek, hay programadores profesionales utilizando Visual Studio y C# para programar. Por lo tanto, en caso de dudas se podría pedir ayuda a estos programadores.

Evidentemente al no tener conocimientos previos en programación con C#, se ha realizado un curso de 30 horas online, de programación con C#.

C# es un lenguaje de programación orientado a objetos desarrollado por Microsoft como parte de la plataforma .NET, y es compatible con nuestro entorno de programación. La sintaxis fue derivada de C y C++ que utiliza el modelo de objetos de la plataforma .NET. Aunque forma parte de la plataforma .NET de Microsoft esta es una API, pero C# es un lenguaje de programación independiente que permite desarrollar aplicaciones para dicha plataforma.

Figura 5-4. Logo C#

5.2.3. Conociendo a nuestro entorno de programación

Figura 5-5. Logo de Visual Studio Fuente: Microsoft

Explorador de soluciones

Una vez creado el proyecto en la ventana Explorador de soluciones (ver **Figura 5-6**), que se encuentra en el lado derecho superior de Visual Studio, se muestran todos los archivos y carpetas del proyecto en una representación gráfica. El Explorador de soluciones es capaz de reconocer la jerarquía del proyecto.

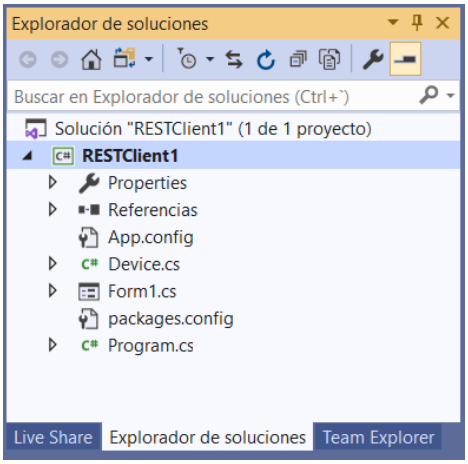


Figura 5-6. Explorador de soluciones.

Un proyecto de Visual Studio se almacena en una solución. Que es simplemente un contenedor con uno o varios proyectos relacionados. A parte del proyecto también guarda la información de compilación, la configuración de las ventanas de Visual Studio y varios archivos más que no están asociados con un proyecto determinado.

Una solución, es un archivo de texto con su propia extensión (.sln), que no está diseñado para modificarse de forma manual.

Editor

En el Editor (ver **Figura 5-7**) se muestra el contenido de los archivos ya sea en forma de texto "Código" o de forma gráfica.

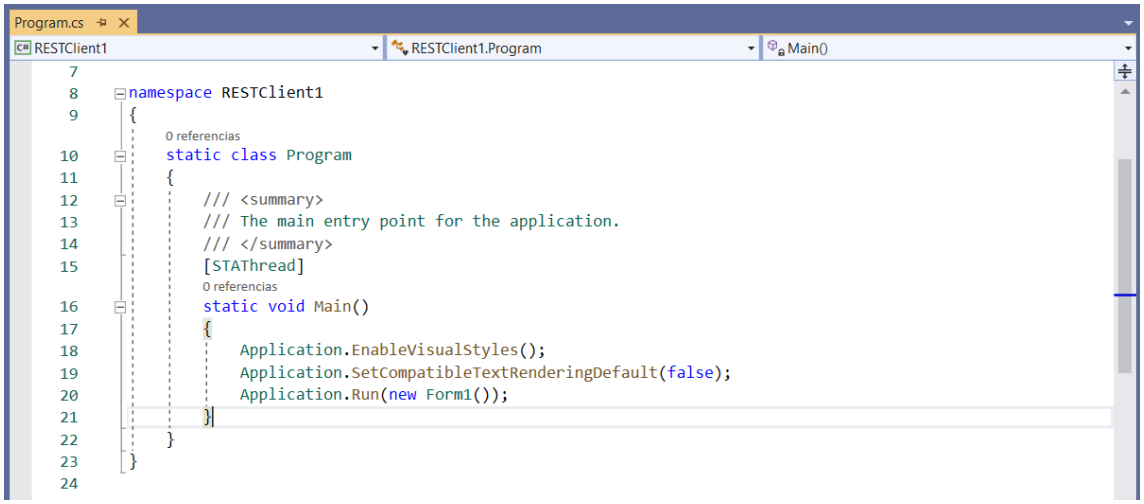


Figura 5-7.Ventana Editor, en Visual Studio.

Menús

Se encuentra en la parte superior de Visual Studio (ver **Figura 5-8**) y contiene los comandos separados por categorías. Por ejemplo, el menú de proyecto tiene agrupados todos los comandos relacionados con el proyecto que está abierto actualmente. Desde este menú podemos agregar nuevos archivos o ver las propiedades del proyecto.

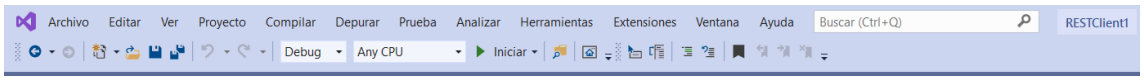


Figura 5-8. Menús en Visual Studio.

Lista de errores

En esta ventana se muestran los errores, (ver **Figura 5-9**) advertencias y mensajes relacionados con el estado actual del código. Visual Studio es capaz de compilar el código automáticamente y mostrar los errores en “tiempo real”. Por ejemplo, si se está escribiendo código en un archivo y le falta un punto y coma o se ha usado una variable sin declarar en la ventana de Lista de errores se mostrará este error.

Si esta ventana no es visible por defecto, se puede habilitar desde la ventana de menús seleccionar Ver y a continuación Lista de errores.

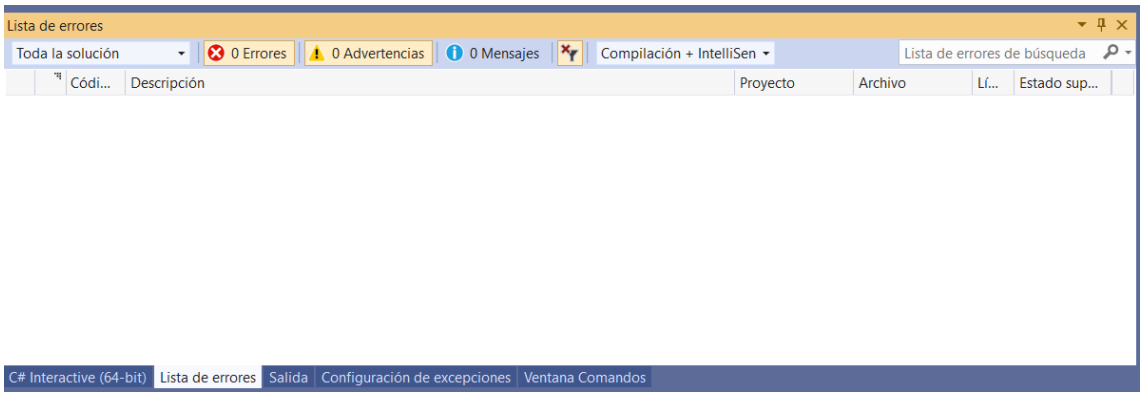


Figura 5-9. Ventana Lista de errores en Visual Studio.

Salida

Una vez compilado el proyecto se muestran los resultados en la ventana de salida (ver **Figura 5-10**). Por ejemplo, si desde el menú Compilar y después compilar, compilamos nuestro primer proyecto, en la ventana de Salida se muestran los resultados, que indican si proyecto fue compilado correctamente, y la cantidad de errores que se han producido en caso de que el resultado no haya sido el correcto.

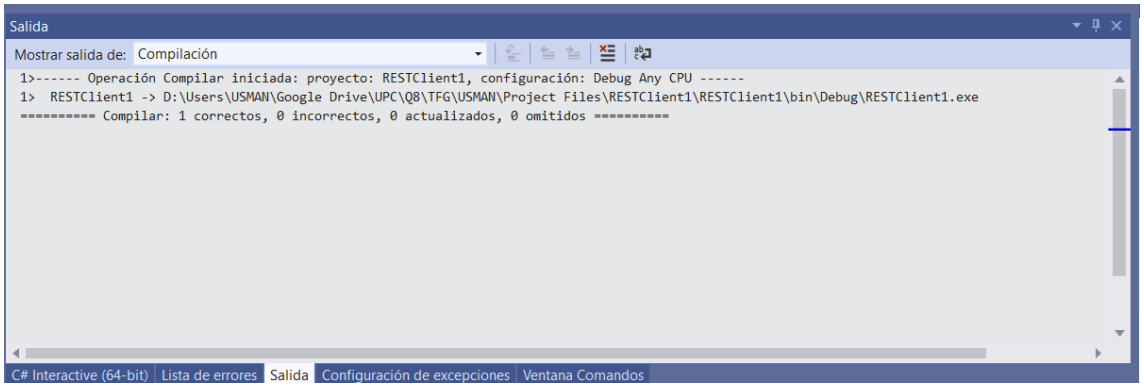


Figura 5-10. Ventana Salida en Visual Studio.

5.3. Primera aplicación de pruebas

Antes de empezar a desarrollar la aplicación de configuración definitiva, para comprobar que realmente nuestra elección de entorno de programación ha sido correcta, se ha decidido crear una primera aplicación “sencilla” de pruebas. Esta aplicación tendrá que cumplir con las funciones básicas que requiere nuestra aplicación de configuración.

Funcionalidades básicas de nuestra aplicación de configuración:

- Comunicar con el servidor REST ”API de configuración ” enviando y recibiendo peticiones.
- Convertir formato de texto tipo JSON en información visual y comprensible para el usuario.
- Programación orientada a objetos para definir objetos que contiene nuestro servidor OPC ya sean los canales , dispositivos, tags, Endpoints, etc.

5.3.1. Crear un nuevo proyecto en Visual Studio

Para empezar a desarrollar la primera aplicación de configuración, se crea un proyecto nuevo (ver **Figura 5-11**) en Visual Studio, eligiendo C# como lenguaje de programación para plataformas .NET y de tipo escritorio.

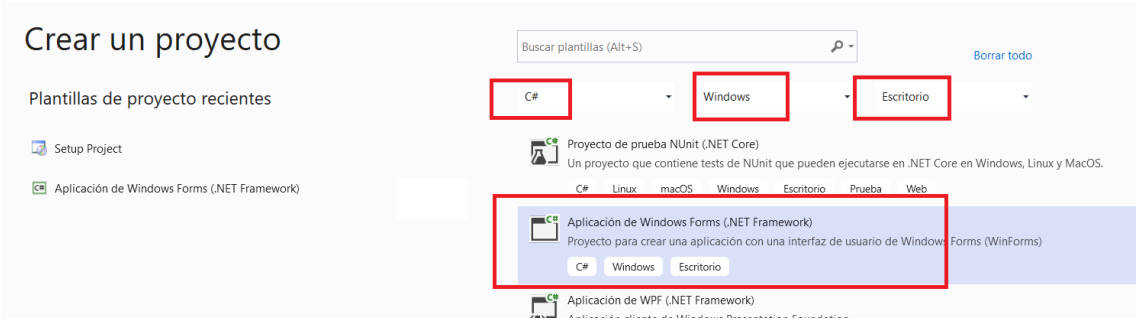


Figura 5-11. Ventana donde se configuran los parámetros de un proyecto a crear.

Tras elegir las características básicas de nuestro proyecto, se pone un nombre al proyecto. En este caso se llama RESTClient1.

Visual Studio se encarga de crear todos los archivos necesarios. Entre los cuales se encuentran los más importantes que son el Program.cs, Form1.cs y App.config.

Si abrimos la clase Program.cs en la ventana Editor (ver **Figura 5-12**).

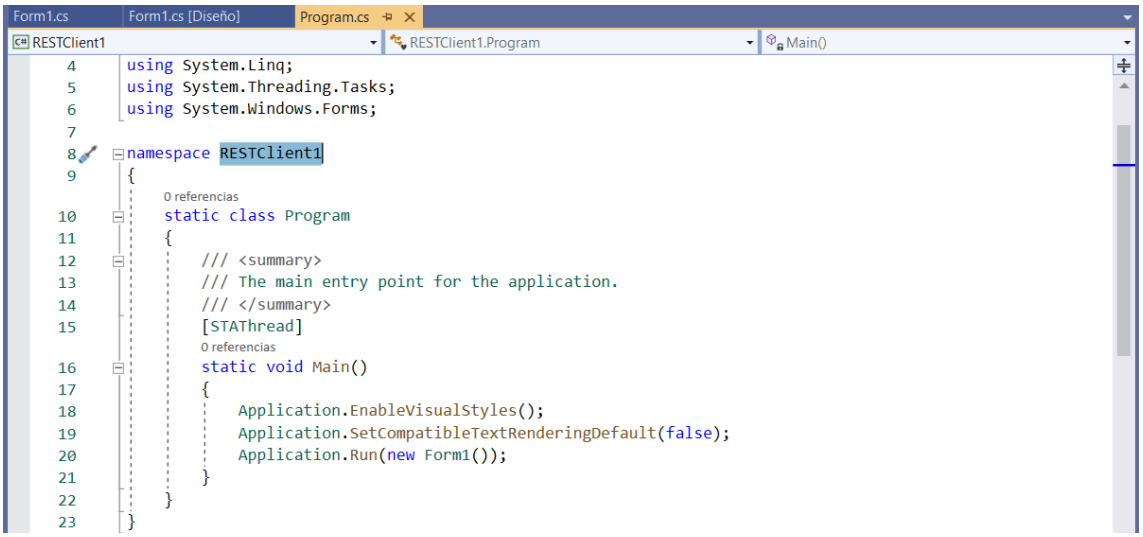


Figura 5-12. La clase Program abierta en la ventana Editor.

Vemos que es la clase principal de nuestro proyecto. Al ejecutar nuestra aplicación el método main() es el primero en invocarse. En C# solo puede haber un punto de entrada en un programa, es decir, un único método main. Si hay más de una clase con el método main el compilador puede lanzar un error sino se ha especificado qué método main es el punto de entrada principal de nuestra aplicación.

El método main contiene varios métodos, de los cuales EnableVisualStyles() y SetCompatibleTextRenderingDefault() son para habilitar los estilos visuales y controles para nuestra aplicación.

El método Run() es el encargado de ejecutar y hacer visible el formulario Form1.

Un formulario es un objeto o control que se visualiza en la pantalla. Sobre el formulario se van a ir colocando otros objetos o controles como cuadro de textos, botones, listas, etc. A su vez también

contiene el código necesario para nuestro programa. Por lo tanto, nos ayudará a construir la interfaz gráfica para el usuario.

5.3.2. Diseño gráfico

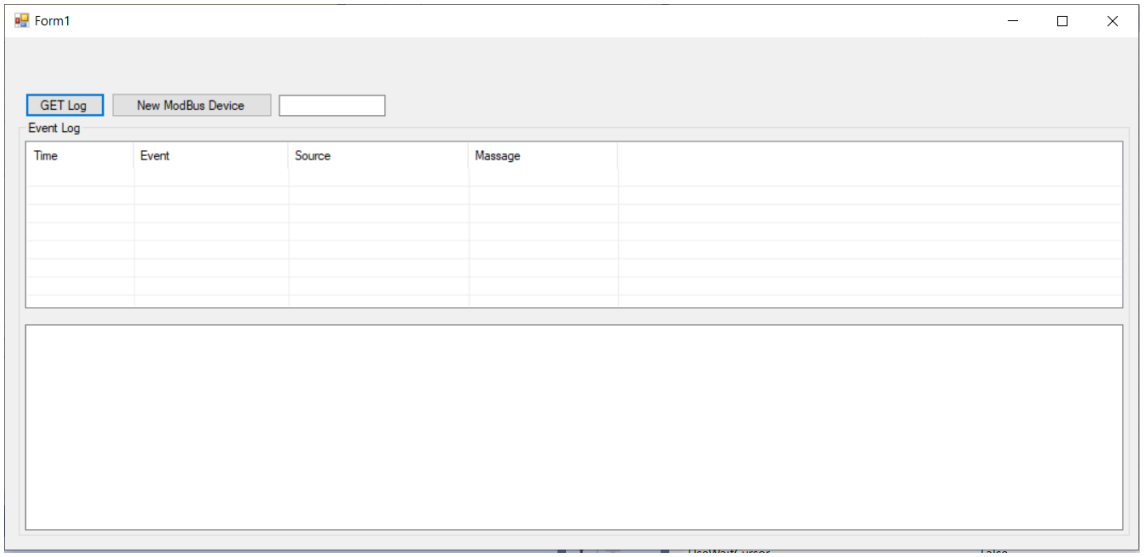


Figura 5-13. Interfaz gráfica de la primera aplicación de pruebas.

Antes de empezar a programar se ha creado la interfaz gráfica de aplicación (ver **Figura 5-13**). Desde la ventana de Cuadro de herramientas (ver **Figura 5-14**), se agregan los siguientes controles:

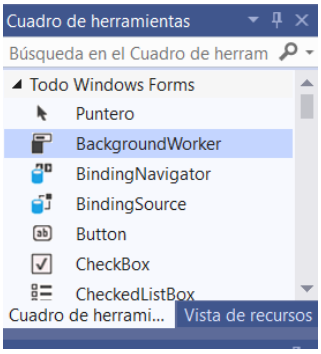


Figura 5-14. Ventana Cuadro de herramientas en Visual Studio.

Botones

GET Log: este botón será el encargado de lanzar una petición de tipo GET a la API de configuración del servidor OPC, para consultar los últimos 100 eventos registrados por el servicio Event Log.

New ModBus Device: lanza una petición de tipo POST para crear un dispositivo ModBus dentro de un canal determinado.

Cuadro de textos

Nombre del dispositivo a crear: este cuadro de texto sirve para que el usuario pueda introducir el nombre del nuevo dispositivo ModBus a crear.

Texto JSON: el cuadro de texto que muestra el JSON que se va a enviar en formato de texto.

ListView

Logger: es una tabla de cuadro de textos, que servirá para mostrar los 100 eventos registrados por el servidor OPC.

5.3.3. Programación

Una vez creada la interfaz gráfica, para que los controles que se han agregado a la interfaz gráfica realicen una función determinada se ha procedido a la programación de estos.

Primeramente, para que nuestra aplicación de pruebas pueda comunicarse con la REST API del servidor OPC, se ha buscado una librería que permite crear un cliente REST.

RestSharp es una de las librerías más populares para plataformas .NET que permite consumir una REST API. Incluye la serialización y deserialización en formatos XML y JSON de forma nativa.

Permite detección automática de tipo de solicitud y respuesta, variedad de autentificaciones y otras características útiles.

Se puede descargar e instalar de una forma sencilla desde el administrador de paquetes de Visual Studio de forma gratuita (ver **Figura 5-15**).

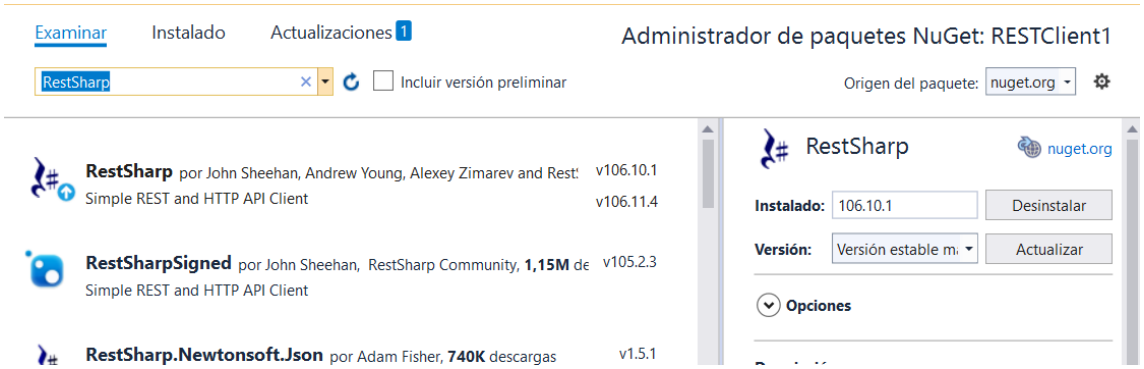


Figura 5-15. Ventana Administrador de paquetes. Donde se ha buscado e instalado el paquete RestSharp.

Una vez instalada la librería se procede a crear un cliente REST para la comunicación con el servidor REST. Para ello se crea una instancia de la clase RestClient indicando la dirección URI o el Endpoint al cual se realizarán las peticiones. Esta **URI o Endpoint base** es común para todas las peticiones que vamos a realizar.

```
var client = new RestClient("https://192.168.1.123:57513/config/v1/");
```

En este caso el cliente REST necesita usuario y contraseña para autentificarse para acceder al servidor REST. Esto se puede hacer llamando al método Authenticator y pasando el usuario y la contraseña de nuestro cliente REST.

```
client.Authenticator = new HttpBasicAuthenticator("Administrator", "*****");
```

Tras crear y definir los parámetros de nuestro cliente REST, ya se puede crear una petición y agregar el cuerpo "body" adecuado.

Primer GET

En este caso cuando el usuario presiona el botón GET Log, se lanza una petición de tipo GET, para consultar los 100 últimos eventos registrados por el servidor OPC.

Para consultar el Event Logger (los últimos 100 eventos), la API de configuración tiene predefinido el siguiente Endpoint:

```
https://192.168.1.123:57513/config/v1/event_log
```

Creando una instancia de la clase RestRequest que también pertenece a la librería RestSharp, se crea una petición de tipo GET, donde también se indica el resto de la dirección URI que se va a agregar a la dirección base ya configurada por el cliente.

```
var request = new RestRequest("event_log", Method.GET);
```

Finalmente, para lanzar la petición se llama al método Execute de nuestro cliente REST, pasándole el request. Una vez que el servidor procese esta petición nos devolverá una respuesta, esta será almacenada en un objeto de tipo IRestResponse.

```
var response = client.Execute(request);
```

La respuesta *response* contiene toda la información respecto nuestra petición lanzada. Por ejemplo, los campos más relevantes son:

Response.Content: contiene la respuesta del servidor en formato de texto JSON.

Response.StatusCode: indica el número de estado de una petición HTTP.

Response.ResponseStatus: indica el estado de la petición.

Dado que la respuesta del servidor OPC es una cadena de caracteres muy larga en formato JSON y esta contiene todos los parámetros "variables" y sus valores correspondientes, la forma más eficiente de mostrar e interactuar con esta información es convirtiéndola en variables separadas. Es decir, crear un objeto que tenga las mismas propiedades que el objeto Event Logger perteneciente al servidor OPC. De manera que todos los parámetros recibidos desde el servidor se almacenan en una instancia del objeto correspondiente en la aplicación de configuración.

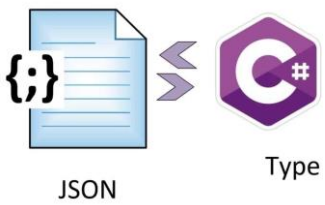


Figura 5-16. Concepto de serialización y deserialización. Fuente: thecodebuzz.com

El proceso de convertir un JSON en una instancia de una clase recibe el nombre de deserialización y serialización si es la operación inversa (ver **Figura 5-16**).

Para convertir la lista de100 eventos recibidos por el servidor OPC, se ha creado una clase denominada EventLogger (ver **Figura 5-17**), que contiene todas las propiedades de un evento. Aunque la librería RestSharp tiene la posibilidad de deserializar o serializar de forma nativa. En este caso se ha decidido utilizar una librería (Newtonsoft) que es más fácil de entender y manejar. Es la misma librería que utiliza RestSharp, para serializar o deserializar.

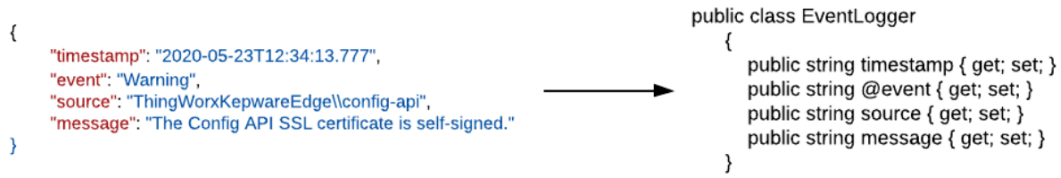


Figura 5-17. De JSON al objeto de tipo EventLogger.

Una vez recibido el *response*, se procede a deserializar esta información. Para ello se crea una lista *eventLogger* de objetos de tipo *EventLogger*. Y el método DeserializeObject se encarga de convertir la cadena de texto JSON y almacenarla en la lista *eventLogger*.

```
eventLogger = JsonConvert.DeserializeObject<List<EventLogger>>(response.Content);
```

Finalmente, se utiliza un bucle para introducir todos los campos de los 100 eventos, en una tabla de textos (ver **Figura 5-18**).

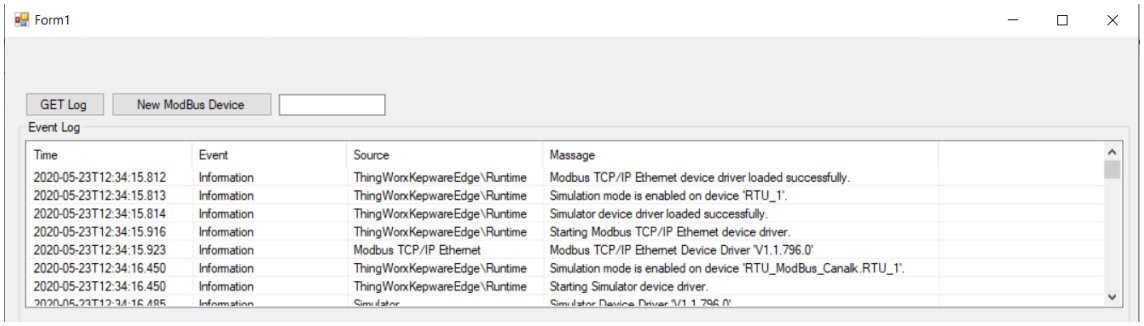


Figura 5-18. Captura de la interfaz gráfica donde se puede ver la tabla que contiene los 100 eventos.

Primer POST

Cuando el usuario hace click sobre el botón New ModBus Device, se ha de lanzar una petición de tipo POST, para crear un nuevo dispositivo en un canal determinado. El nombre del dispositivo a crear por defecto tomará como nombre New ModBus Device si el usuario no introduce ningún otro en el campo indicado (ver **Figura 5-19**).

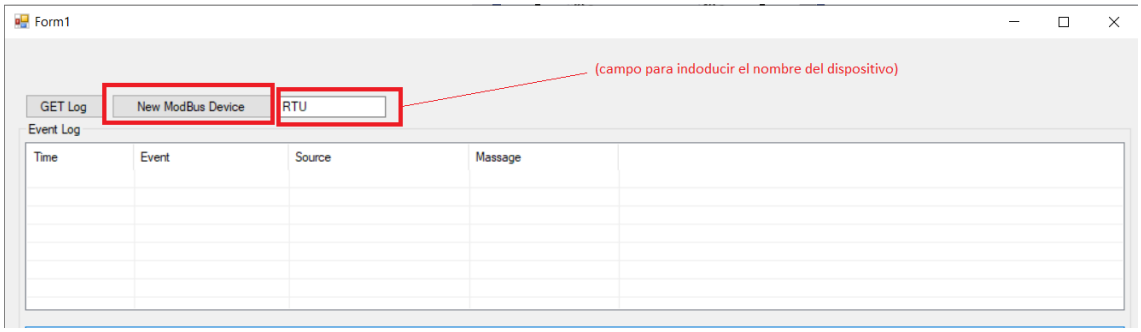


Figura 5-19. Captura que muestra donde está el botón New ModBus Device y el cuadro de texto para introducir el nombre del dispositivo.

Para programar esta acción, se vuelve a utilizar el mismo cliente REST, con la misma autentificación y la URI base. Pero la petición será diferente, ya que es de tipo POST y la parte del Endpoint que se va a sumar la URI base también cambia. De manera que el objeto request a enviar será:

```
Var request =new RestRequest("project/channels/RTU_ModBus_Canal/devices", Method.POST);
```

En este caso tal y como se puede observar en el Endpoint, el dispositivo se va a crear dentro de un canal llamado RTU_ModBus_Canal. Este canal ya fue creado previamente en este proyecto.

Después de crear el request, se deberá incluir el body correspondiente a esta petición. Para ello, primeramente, se ha creado una nueva clase denominada Device. En esta clase se han se han definido todas las propiedades que puede tener un dispositivo ModBus.

Para que la librería Newtonsoft pueda serializar o deserializar correctamente un JSON, el nombre de todas las propiedades de nuestra clase y el nombre de las propiedades incluidas en el JSON deben de ser idénticos.

Por ejemplo, si en el JSON existe una variable booleana que se llama “Name”, en la clase correspondiente se deberá declarar una variable booleana que se llama exactamente igual “Name”.

Dependiendo del tipo de driver a utilizar, un dispositivo o canal puede tener más de 30 propiedades. En algunas de estas propiedades el nombre de la propiedad incluye un (.). Esto está definido por el fabricante en la API de Configuración, de manera que no se puede cambiar.

Ejemplo:

```
common.ALLTYPES_NAME
servermain.MULTIPLE_TYPES_DEVICE_DRIVER
servermain.DEVICE_CHANNEL_ASSIGNMENT
```

Esto es un **problema** para declarar todas estas propiedades en nuestra clase. Ya que en C# no se puede declarar variables que incluyen un punto en el nombre.

Para solucionar este problema se ha creado un vínculo entre el nombre de la propiedad declarada en la clase y el nombre que se va a utilizar para convertir el JSON. Esto se puede hacer utilizando el atributo [JsonProperty] especificando el nombre de la propiedad que va a utilizar la librería, antes de declarar cada propiedad en la clase.

Ejemplo:

```
[JsonProperty("common.ALLTYPES_NAME")]
public string CommonAlltypesName { get; set; }
[JsonProperty("common.ALLTYPES_DESCRIPTION")]
public string CommonAlltypesDescription { get; set; }
```

Tras solucionar este problema, se crea una instancia del objeto Device denominada device. Se asignan los valores correspondientes, que son obligatorios para crear un dispositivo. A continuación, se convierte este objeto de tipo Device, en una estructura de texto tipo JSON utilizando el método SerializeObject.

```
string newDevicebody = JsonConvert.SerializeObject(device, Formatting.Indented);
```

Una forma de comprobar que la operación de serializar se ha realizado correctamente antes de enviar la cadena de texto convertida al servidor, esta se imprime en un cuadro de texto en la interfaz gráfica (ver Figura 5-20).

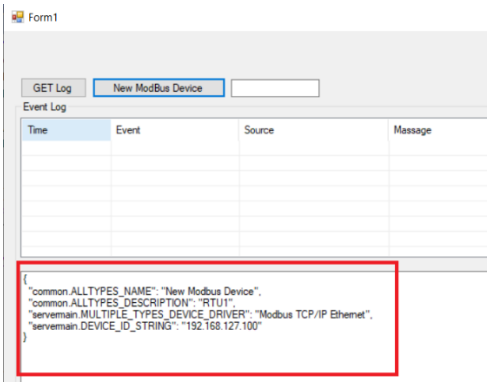


Figura 5-20. La captura que muestra el cuadro de texto, donde se ha imprimido el JSON antes de enviarlo.

Se incluye esta cadena de texto al request a enviar:

```
request.AddJsonBody(newDevicebody);
```

Finalmente se envía esta petición al servidor. Y la respuesta de esta se recibe en un objeto de tipo IRestResponse. El contenido de la respuesta se imprime en el mismo cuadro de texto que sirve para visualizar el JSON.

Imprimir la respuesta en formato de texto puede servir para comprar si el dispositivo fue creado correctamente o ha habido algún problema (ver Figura 5-21). Por ejemplo, en el siguiente caso se ha intentado crear un dispositivo que ya existe en este canal.

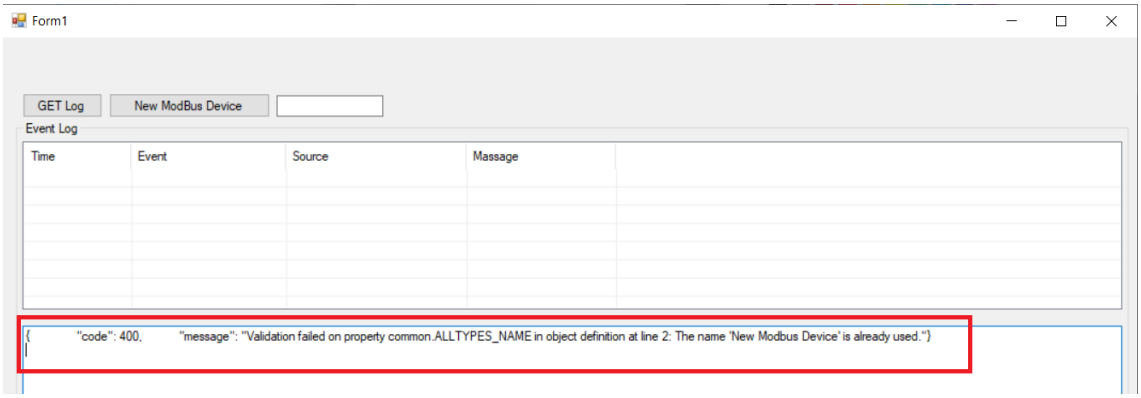


Figura 5-21. La respuesta del servidor cuando se intenta crear un dispositivo que ya existe.

Diagrama de fujo

Para una mejor comprensión del código, se han realizado dos diagramas de flujo (ver **Figura 5-22**). Cada uno muestra el flujo de ejecución del código asociado del botón correspondiente.

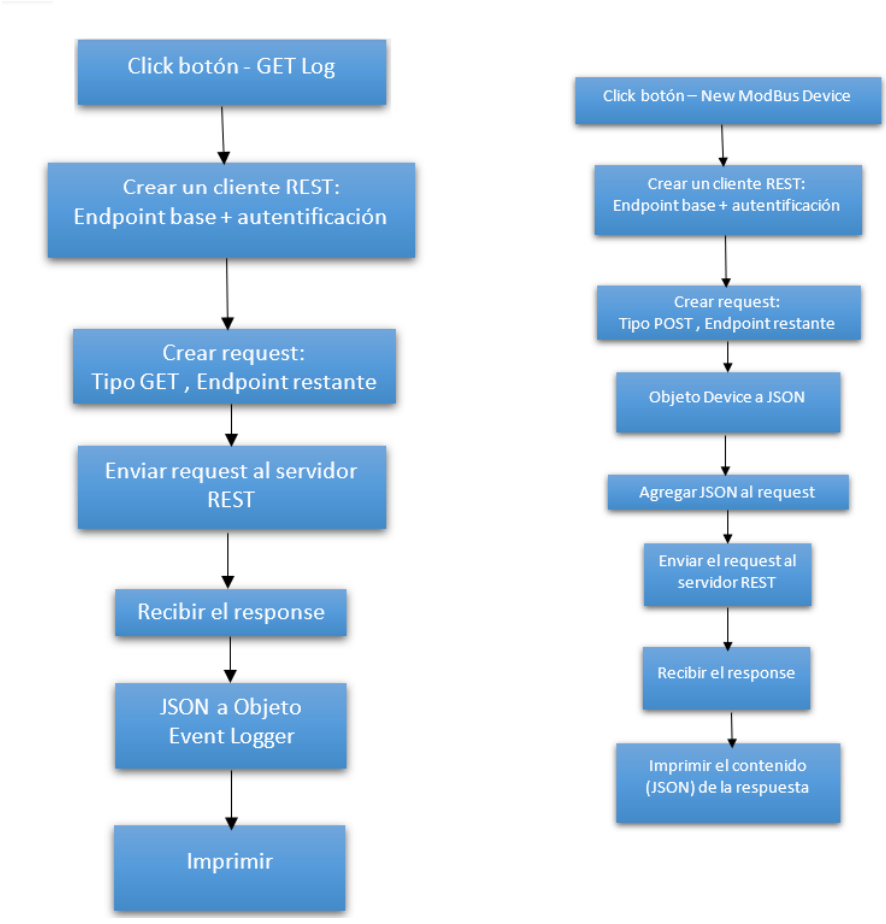


Figura 5-22. Diagrama de flujo que explica la lógica que se sigue durante proceso de creación del dispositivo y lectura de los 100 eventos.

5.3.4. Conclusiones

Tras crear la primera aplicación de pruebas, se concluye que la IDE y el lenguaje de programación están a la altura para empezar a programar la aplicación de configuración definitiva. Dado que cumple con los objetivos y funcionalidades básicas que se buscaban.

En la siguiente tabla (ver **Tabla 5-1**) se muestran las funcionalidades básicas que se buscaban y como se han conseguido estos objetivos marcados.

Tabla 5-1. Resumen de cómo se han cumplido los objetivos marcados.

Funcionalidades básicas	Solución
Comunicar con el servidor REST ”API de configuración ” enviando y recibiendo peticiones.	Esto se ha conseguido con la ayuda de la librería RestSharp. Ya que permite: • Crear un cliente REST. • Comunicación vía HTTP. • Enviar y recibir todo tipo de peticiones.
Convertir formato de texto tipo JSON en información visual y comprensible para el usuario.	La librería Newtonsoft permite serializar y deserializar cadenas de texto de tipo JSON de una forma sencilla y eficaz.
Programación orientada a objetos para definir objetos que contiene nuestro servidor OPC, canales, dispositivos, tags, Endpoints, etc.	C# es un lenguaje de programación orientado a objetos.

5.4. Aplicación de configuración final

5.4.1. Requisitos

El departamento de soporte técnico de Logitek S.A, continuamente recibe peticiones de mejoras para la interfaz nativa de KEPServerEX. Los clientes envían estas peticiones para mejorar la forma de interactuar con la interfaz y también incluir nuevas funciones. Por lo tanto, antes de empezar a diseñar la aplicación de configuración final, se han tenido en cuenta estas mejoras. A parte de las peticiones de mejoras enviadas por los clientes, también desde el departamento de soporte técnico y Solution Managers se han planteado varios requisitos que deberá cumplir nuestra aplicación de configuración.

Requisitos y peticiones:

- La aplicación de configuración debe permitir al usuario, consultar la configuración actual del proyecto que hay cargado en el Runtime. No solo consultar sino, borrar, modificar e incluso deberá permitir crear nuevos canales, dispositivos y tags.
- Debe permitir consultar, borrar o crear los OPC UA Endpoints.
- Consultar el Logger del servidor OPC, filtrando por el tipo de evento o incluso filtrar eventos por fecha y hora.
- La aplicación de configuración debe permitir consultar Loggers de varios servidores a la vez en la misma ventana.

- Exportar el Logger en un archivo de texto.
- Si hay varios views en la ventana principal, hay que permitir al usuario esconder o mostrar uno o varios views. De manera que el usuario solo ve la información de mayor interés en la ventana principal.
- KEPServerEX muestra todas las propiedades de un tag en la ventana principal. Pero para modificar alguna propiedad hay que dar doble click y desde una ventana secundaria se modificarán estas propiedades. Nuestra aplicación de configuración deberá mostrar todas las propiedades en la ventana principal, pero también modificar una o varias propiedades desde la misma ventana.
- Crear un canal o dispositivo nuevo, a partir de copiar configuración de un canal o dispositivo ya creado en el servidor.
- Exportar e importar tags.
- La aplicación de configuración debería tener un aspecto diferente que una típica aplicación creada en Visual Studio. Es decir, cambiar el estilo que viene por defecto de los formularios y controles para dar un aspecto más moderno y profesional.

Teniendo en cuenta los requisitos, se ha procedido a diseñar la interfaz gráfica de la aplicación de configuración. En el siguiente apartado se explicará cómo se ha construido la interfaz en base a los criterios y requisitos anteriores, desde un punto de vista de usuario final. De manera que se ve, como el usuario navegaría por la interfaz gráfica que tiene todas las funcionalidades y requerimientos planteados.

5.4.2. Diseño de la interfaz gráfica

Para construir la aplicación de configuración definitiva, primeramente, teniendo en cuenta las funcionalidades y necesidades planteadas, se ha comenzado a diseñar la interfaz gráfica de la aplicación.

Para dar un estilo más moderno y profesional (ver **Figura 5-23**) se ha decidido cambiar el estilo visual de los formularios y los componentes que vienen por defecto en Visual Studio. Para ello se ha instalado un paquete de librerías ***MaterialSkin*** para plataformas .NET.

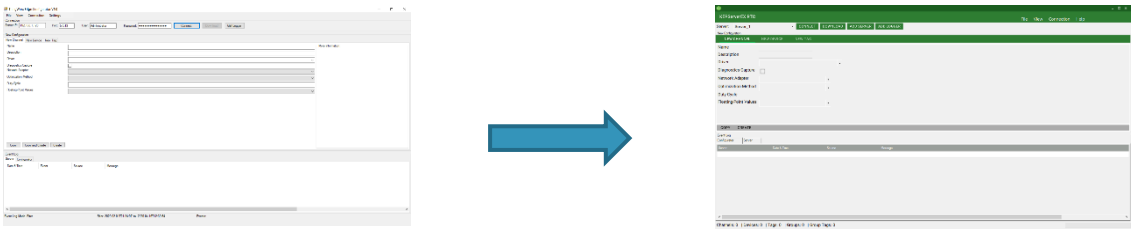


Figura 5-23. El proceso de cambio de estilo de los formularios (ventanas).

La aplicación de configuración constará de una ventana principal y varias ventanas secundarias. La ventana principal se ha dividido en 6 secciones diferentes. Para comprender mejor el contenido de la ventana principal, primeramente, se explicará el contenido y objetivo de las ventanas secundarias.

5.4.2.1. Ventanas secundarias

Para no sobrecargar la ventana principal, se han creado varias ventanas secundarias que permiten realizar diferentes funciones. Por ejemplo, agregar y/o modificar usuarios, abrir un nuevo Event Logger, crear un nuevo grupo de tags o agregar y/o modificar los OPC UA Endpoints.

Server Connections Manager

Para la comunicación con la REST API del servidor OPC, el usuario debe tener la dirección IP, el puerto, especificar el protocolo (http o https) a utilizar y el nombre del usuario con contraseña en caso de que sean necesarios para la autentificación.

Todos estos parámetros se han encapsulado (ver **Tabla 5-2**) en un único concepto llamado conexión. Por lo tanto, el usuario en aplicación de configuración debe crear una conexión para comunicar con un servidor OPC determinado.

Tabla 5-2. Las propiedades que puede tener una conexión.

Propiedad	Descripción
Nombre	El nombre que identifica esta conexión.
Descripción	Una breve descripción para esta conexión.
IP	IP del servidor OPC.
Puerto	Puerto del servidor OPC.
Https	Un check que permite especificar el protocolo a utilizar.
Usuario	El nombre del usuario con el cual se va a realizar la autentificación.
Contraseña	Contraseña con la cual se va a realizar la autentificación.

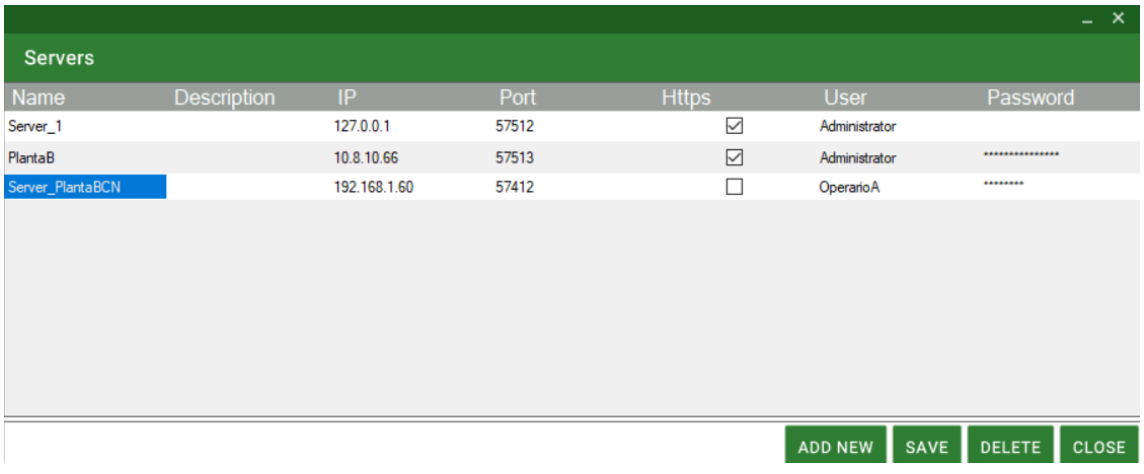


Figura 5-24. Captura de la ventana Server Connections Manager.

Dado que una empresa puede tener varios servidores OPC (ver **Figura 5-24**), instalados en una misma planta, se ha decidido crear una ventana para guardar y/o modificar varias conexiones. Los nombres de todas estas conexiones se mostrarán en la ventana principal, en la sección de connection en forma de desplegable. De manera que, si el usuario desea configurar un servidor OPC, simplemente deberá seleccionar el nombre de la conexión correspondiente y conectar con dicho servidor OPC.

OPC UA Endpoints

Desde esta ventana (ver **Figura 5-25**) el usuario puede consultar, modificar, borrar o crear los OPC UA Endpoints. Se mostrarán todos los OPC UA Endpoints en forma de lista, de manera que el usuario selecciona un OPC UA Endpoint y ve todas sus propiedades.

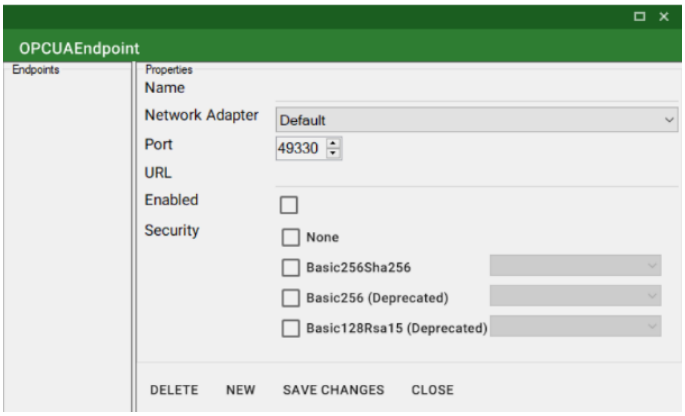


Figura 5-25. Ventana secundaria OPCUAEndpoint.

Server Event Logger

Es una ventana (ver **Figura 5-26**) secundaria que se incrusta en la ventana principal y también en otras ventanas secundarias. En esta ventana secundaria se puede interactuar con Event Logger del servidor OPC, de manera que se muestran todos los eventos seleccionados por el usuario en una tabla. También existe la posibilidad de guardar esta tabla en un archivo de texto en formato txt.

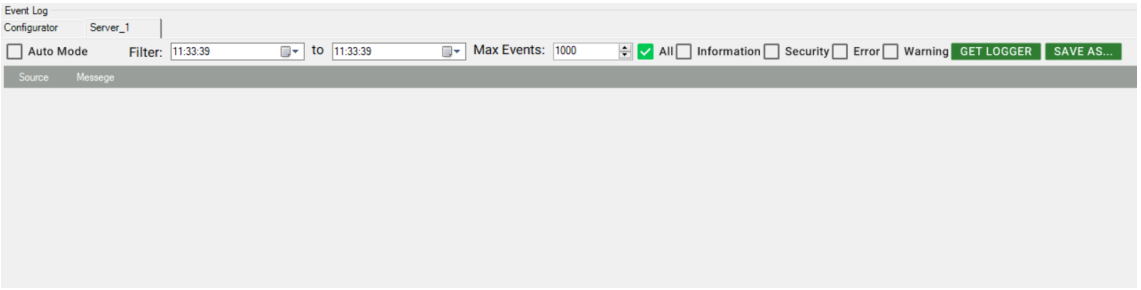


Figura 5-26. Captura de la ventana secundaria Server Event Logger.

Existen dos modos de carga de eventos, que son:

Auto mode: en este modo, se cargan los últimos 100 eventos registrados por el servidor OPC, de forma periódica. Se puede seleccionar este modo marcando el check Auto Mode. Una vez seleccionado este modo el usuario puede seleccionar con qué frecuencia se muestran los eventos.

Filtered: desmarcando el check Auto Mode, se selecciona el modo filtrado. En este modo el usuario tiene la posibilidad de cargar los eventos filtrando por días y fechas. También existe la posibilidad de filtrar por el tipo de evento.

Multi Event Logger

Desde esta ventana (ver **Figura 5-27**) se podrán consultar dos Server Event Logger a la vez. El usuario mediante un desplegable podrá ver el nombre de todas las conexiones guardadas y, seleccionando una conexión determinada, podrá ver el Event Logger de ese servidor OPC.

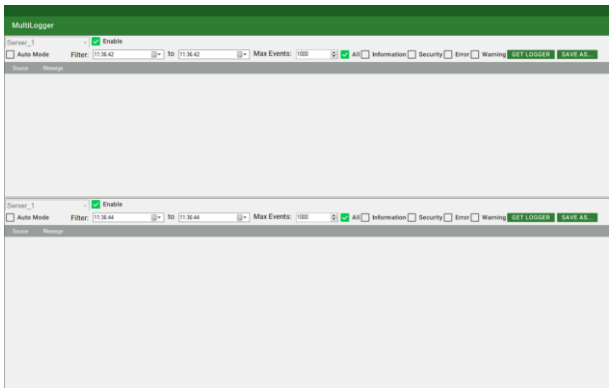


Figura 5-27. Captura de la ventana secundaria Multi Event Logger.

New Tag Group Creator

Sirve para crear un nuevo grupo de tags. El usuario deberá introducir el nombre y una breve descripción en un formulario. Una vez introducidos estos datos, el usuario mediante un botón enviará una petición al servidor, para crear dicho grupo.

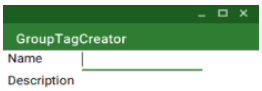


Figura 5-28. Ventana secundaria GroupTagCreator.

About

Ventana que mostrará información relevante sobre la aplicación de configuración. Por ejemplo, el nombre completo de la aplicación, la versión y el logo. A su vez también se podrá consultar el contacto del desarrollador para informar de los bugs, peticiones de mejoras o ponerse en contacto con el soporte técnico en caso de necesitar ayuda técnica.

Figura 5-29.Captura de la ventana secundaria About.

5.4.2.2. Ventana Principal

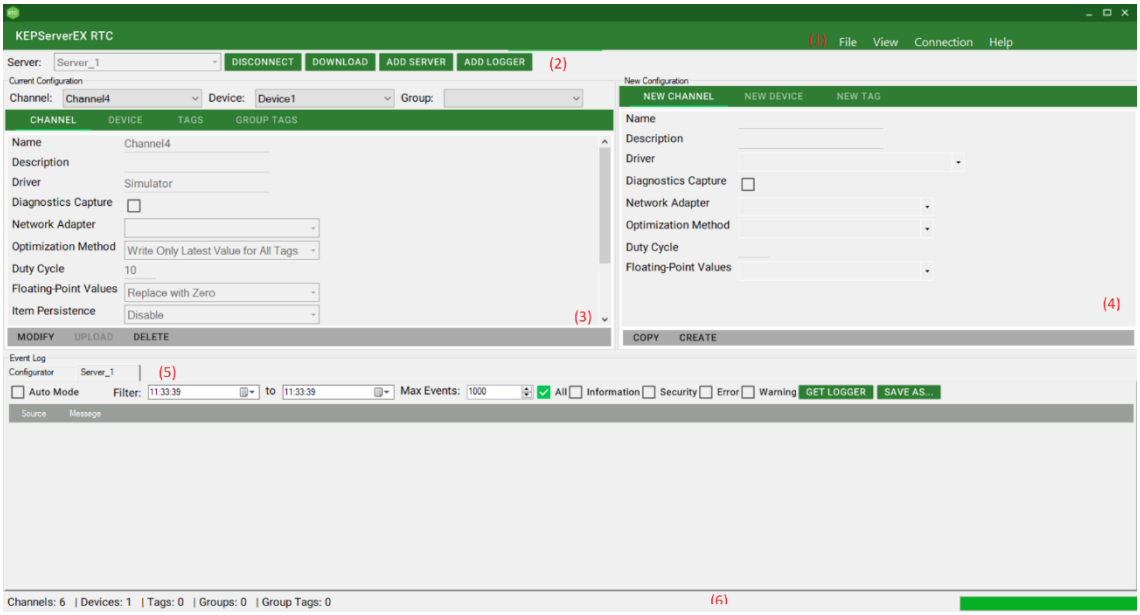


Figura 5-30. Captura de la ventana principal.

Barra de menús

La barra de menús contiene varias opciones y herramientas en forma de menús desplegables. De esta forma el usuario puede interactuar con las ventanas secundarias, opciones y funcionalidades de la aplicación de configuración.

La barra de menús contiene 4 grupos principales: (ver Tabla 5-3)

Tabla 5-3. Todos los menús y submenús que tiene la barra de menús. A su vez, en esta tabla, se explica la funcionalidad de cada menú.

Grupo	Descripción
File	Contiene un submenú llamado Exit, el cual sirve para salir de la aplicación.
View	Menú para visualizar o esconder una o varias de las secciones de la ventana principal. Submenús: • Connection • Current Configuration • New Configuration • Event Log
Connection	Grupo que contiene varios submenús relacionados con las conexiones, tanto con el servidor OPC o Servidor REST. Submenús: • Connect: acceso rápido al botón conectar. • Edit servers connections: acceso rápido para acceder a la ventana

	secundaria *Server connections manager*. • MultiLogger: acceso rápido para acceder a la ventana secundaria *MultiLogger*. • OPC UA Endpoints: acceso rápido para acceder a la ventana secundaria *OPC UA Endpoints.*
Help	Contiene varios submenús para proporcionar información de ayuda al usuario.

Connection

En esta sección (ver **Figura 5-31**) el usuario puede realizar pruebas de comunicación con el servidor REST, descargar la información actual que este contiene, agregar varios Event Logger en la sección de Event Log y acceder a la ventana secundaria de *Server connections manager.*

Figura 5-31. Captura de la sección Connection.

La sección contiene un desplegable con los nombres de todas las conexiones guardadas por el usuario, para que este pueda seleccionar una conexión y conectar con ese servidor REST determinado.

Current Configuration

La sección Current Configuration mostrará la configuración actual del servidor OPC. Por ejemplo, propiedades de todos los canales, dispositivos y tags ya creados.

A parte de mostrar la configuración actual, también permite modificar uno o varios parámetros de un objeto determinado. Por ejemplo, cambiar el nombre de un canal ya existente, cambiar la IP de un dispositivo o borrar un tag.

Tal y como se puede observar en las siguientes capturas de pantalla (ver **Figura 5-32** y **Figura 5-33**), los nombres de todos los canales ya creados en el servidor OPC se mostrarán en una lista desplegable (1). Todas las propiedades del canal seleccionado en el desplegable se mostrarán en la pestaña Chanel(2). Los nombres de todos los dispositivos pertenecientes al canal seleccionado también se mostrarán en forma de lista en un desplegable (3). La información del dispositivo seleccionado se muestra en la pestaña Device (4). Si existen grupos de tags en el dispositivo seleccionado, los nombres de los grupos de tags se muestran en el desplegable (5).

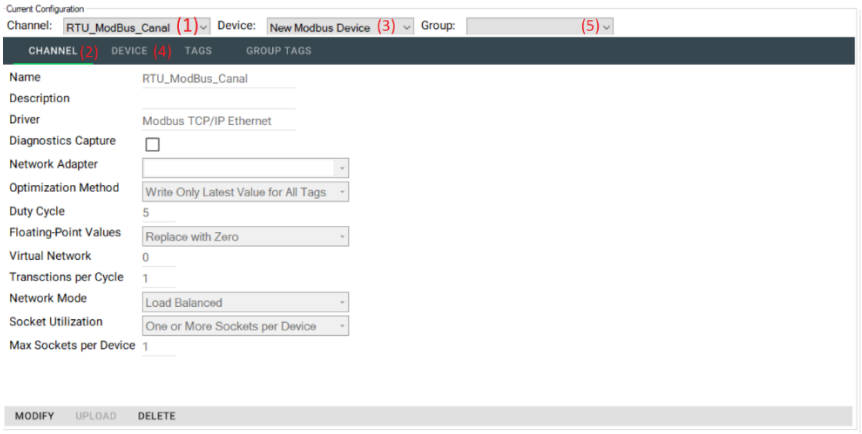


Figura 5-32. Captura de la sección Configuración actual.

Todos los tags (sin grupo) que contiene el dispositivo seleccionado se muestran una tabla en la pestaña “TAGS”. Cada columna de la tabla es una propiedad de un tag determinado. De la misma manera en la pestaña GROUP TAGS (6), se muestran los tags pertenecientes al grupo de tags seleccionados en el desplegable (5).

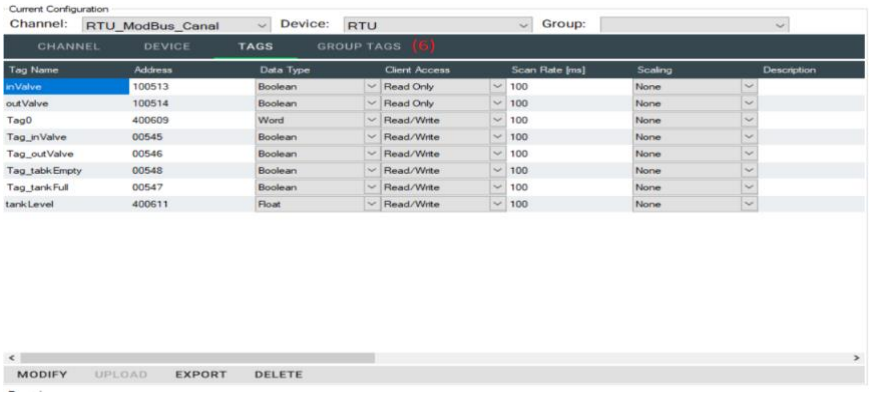


Figura 5-33.Captura de la sección Configuración actual, donde se muestra la pestaña TAGS.

En las pestañas TAGS y GROUP TAGS, aparte de modificar y borrar los tags, también existe otro botón llamado EXPORT. Este botón permite al usuario exportar todos los tags mostrados en un archivo, en formato de texto tipo JSON.

New Configuration

Desde esta sección, se podrá introducir y crear una nueva configuración. Es decir, crear un nuevo canal, un nuevo dispositivo, uno o varios tags a la vez y crear un nuevo grupo de tags.

Esta sección (ver **Figura 5-34**) dispone de 3 pestañas y es muy similar a la Current Configuration. La pestaña NEW CHANNEL (1), permite crear un nuevo canal en el servidor OPC al cual el usuario está conectado actualmente. Dependiendo del tipo de driver seleccionado las propiedades del canal se cambiarán automáticamente. Por ejemplo, un canal de tipo ModBus, posee muchos más parámetros a configurar que un canal tipo Simulator.

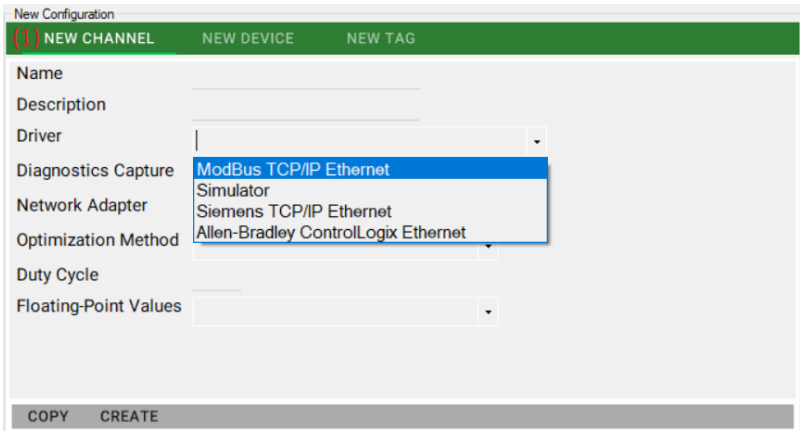


Figura 5-34. Captura de pantalla de la sección New Configuration.

Una vez introducidos todos los parámetros más relevantes y obligatorios, el usuario puede crear este canal dando click sobre el botón CREATE.

En esta pestaña también existe la posibilidad de copiar un canal ya creado, a través del botón COPY. De manera que, si el usuario hace click sobre el botón COPY y existe algún canal actualmente cargado en la ventana Current Configuration, ese será copiado en la pestaña NEW CHANNEL. Una vez copiado el canal, el usuario puede modificar las propiedades copiadas y a posteriori crear este nuevo canal dando click sobre el botón CREATE.

Tras haber creado el canal, el usuario deberá ir a la pestaña NEW DEVICE (2), para crear un nuevo dispositivo perteneciente a este canal que se acaba de crear. De forma automática el canal selecionado en esta pestaña será el que fue creado en el paso anterior. Pero el usuario tambien puede seleccionar cualquier otro canal ya creado en el servidor OPC, filtrando por el tipo de driver.

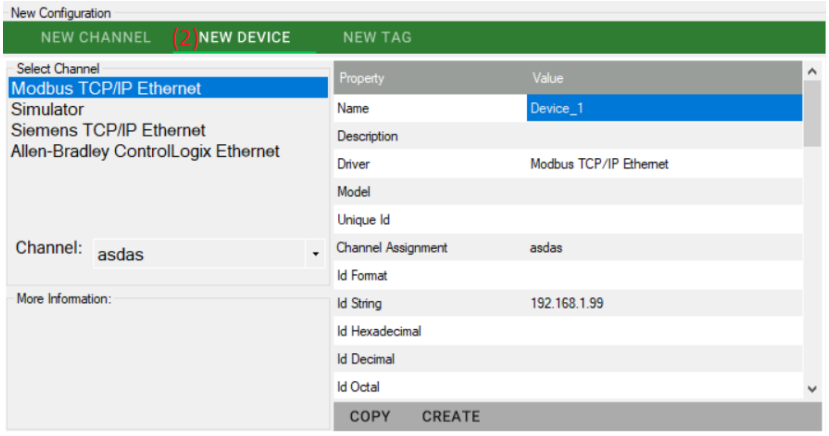


Figura 5-35. Captura de pantalla donde se muestra la pestaña NEW DEVICE, dentro de la sección New Configuration.

Por ejemplo, si el usuario seleciona el driver ModBus TCP/IP Ethernet (ver **Figura 5-35**), a continuación, en el desplegable Channel, se cargarán en forma de lista todos los canales de tipo ModBus ya creados. Una vez selecionado el canal, el programa cargará todas las propiedades de este tipo de dispositivo. El usuario deberá introducir valores a todas las propiedades obligatorias para crear un dispositivo nuevo, y finalmente click sobre el boton CREATE.

Finalmente, para crear tags, el usuario se dirigirá a la pestaña NEW TAG (3) (ver **Figura 5-36**). En esta pestaña también se selecciona el canal y dispositivo en el cual se desean crear los tags. Por defecto los tags no pertenecen a ningún grupo de tags en concreto, pero el usuario puede elegir un grupo de tags desde el desplegable Grupo. Si no existe ningún grupo en este dispositivo, dando click sobre el botón NEW GROUP se abre la ventana secundaria New Tag Group Creator, la cual permite crear un nuevo grupo de tags.

Tras seleccionar el canal y el dispositivo (ver **Figura 5-36**) el usuario ya puede comenzar a agregar los tags, dando click sobre el botón ADD. El usuario puede crear tantos tags como desee, e ir dando clicks sobre este botón.

Los tags ya tienen un valor por defecto a la hora de crear. Pero estos valores se pueden modificar por el usuario.

La ventana TAG de la sección Current Configuration permite al usuario exportar tags ya creados en el servidor OPC. En la ventana NEW TAG existe la posibilidad de importar un archivo de texto que contiene los tags en formato JSON.

Una vez definidos todos los tags necesarios, el usuario dará click sobre CREATE, para enviar toda esta información al servidor OPC.

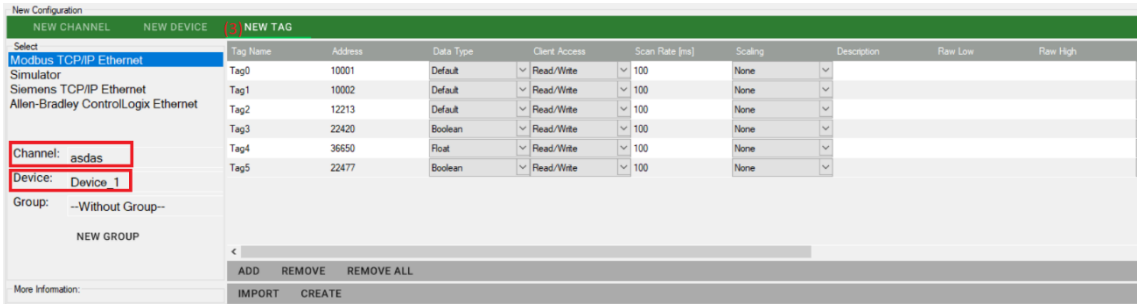


Figura 5-36. Canal y dispositivo seleccionados por el usuario en la sección New Configuration, dentro de la pestaña NEW TAG.

Event Log

Esta sección contiene dos pestañas fijadas (ver **Figura 5-37**), que son el Configurator y el Server Logger. el nombre de la pestaña Server Logger depende de la conexión seleccionada. Ya que el nombre de esta pestaña será el mismo que el nombre de la conexión.

Figura 5-37. Captura de la sección Event Log, donde se muestran las dos pestañas prefijadas.

La pestaña configurator contiene una tabla, donde se registran todos los eventos, excepciones y advertencias producidas por la propia aplicación de configuración.

Por otro lado, cuando se conecta a la API de configuración del servidor OPC, seleccionando el nombre de una conexión desde la sección de Connection, dentro de la pestaña Server Logger se mostrará una ventana secundaria de tipo Server Event Logger.

A parte de estas dos pestañas el usuario puede agregar más pestañas, en las que estará incrustada una ventana secundaria de tipo Server Event Logger. Estas nuevas pestañas agregadas no son fijas, de manera que el usuario las puede borrar mediante un botón (ver **Figura 5-38**).

Figura 5-38. Captura de pantalla que muestra donde se encuentra el nuevo Server Event Logger agregado.

Para agregar un nuevo Server Event Logger, desde la sección de Connection, el usuario da un click al botón “ADD LOGGER”, le aparecerá una lista con todos los nombres de las conexiones guardadas (ver **Figura 5-39**). Si se selecciona un nombre determinado se agregará una nueva pestaña de tipo Server Event Logger, en la sección de Event Log.

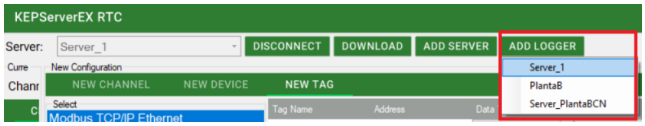


Figura 5-39. Menú que muestra la lista de todas las conexiones guardadas.

Barra de estado

Muestra una barra de progreso (ver **Figura 5-40**) que sirve para ver si la aplicación de configuración está realizando alguna tarea. También contiene varios contadores que indican la cantidad de canales, dispositivos, tags, grupos y tags en los grupos que hay actualmente en el servidor.

Figura 5-40. Barra de estado.

5.4.3. Programación

Una vez creada la interfaz gráfica en el Visual Studio, se procede a empezar a programar. Dado que la idea es replicar en nuestra aplicación casi todos los objetos que hay en el servidor OPC, se procede a crear las clases necesarias para definir las propiedades y funcionalidades de estos objetos. La clase principal y la más importante es la clase Main, que es la encargada de proporcionar todas las funcionalidades de nuestra aplicación con la ayuda de todas las otras clases creadas.

Tabla 5-4. Todas las clases creadas en el proyecto.

Tipo	Clase
Clase Simple	• Channel • ModBusDevice • SimulatorDevice • SiemensDevice • AllenDevice • Tag • TagGroup • EventLogger • Response • ServerProperties • UAEndpoint • ConfiguratorEvent • Server
Clase asociada a una ventana	• Servers • Main • ServerLogger • MultiLogger • OPCUAEndpoint • About • GroupTagCreator

En este proyecto se han creado un total de 20 clases, que interactúan entre ellas para conseguir las funcionalidades requeridas en la aplicación de configuración (ver **Tabla 5-4**). Las clases están divididas en dos grupos que son, las clases simples y las clases asociadas a las ventanas.

En el diagrama de flujo (ver **Figura 5-41**) se observa un ejemplo de la topología utilizada para programar la aplicación. La clase Main es una clase asociada a la ventana principal, se encarga de comunicar con varias clases simples y todas las clases asociadas a las ventanas, para intercambiar información enviando y/o recibiendo objetos a sus métodos correspondientes. Por ejemplo, para mostrar todos los canales ya creados en el servidor OPC, la clase Main se comunica con la clase Server pasando una instancia de la clase ServerProperties. Con esta información la clase Server se comunica con el servidor OPC mediante un cliente REST para consultar todos los canales. Una vez obtenidos los canales en formato JSON los convierte en una lista de objetos de tipo Channel y finalmente devuelve esta lista a la clase Main.

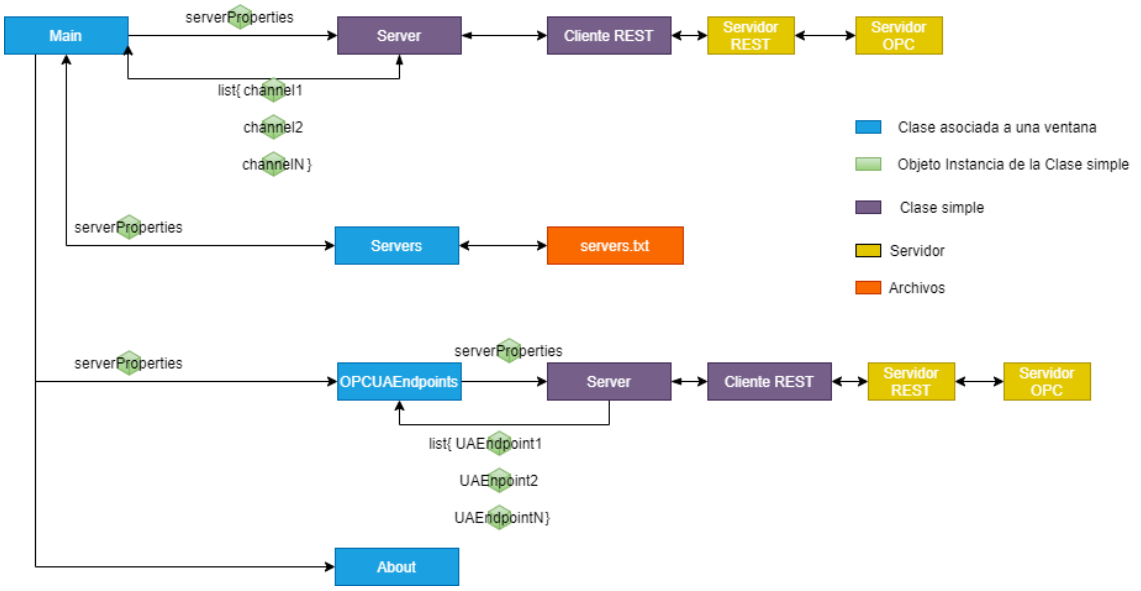


Figura 5-41. Un ejemplo de la topología utilizada para la programación de la aplicación.

El objetivo de este apartado no es explicar o mostrar el código escrito para implementar todas las funcionalidades explicadas en el apartado de diseño. Por lo tanto, se utilizarán capturas de pantallas y diagramas de flujo para explicar la lógica programada.

Para trabajar de forma cómoda y con una mejor organización, el proyecto se ha dividido en varias clases separadas (archivos diferentes). A continuación, se hablará de todas las clases creadas y la función de cada una.

5.4.3.1. Clases simples

Podemos diferenciar las clases en dos tipos bien diferenciables. Las clases simples y las que están asociadas a una ventana. Las clases simples se han creado para definir las propiedades (ver Anexo V) de los objetos a replicar y, también, algunas clases simples tiene métodos y funciones propias para realizar una función determinada.

Channel

Clase creada para definir todas las propiedades que puede tener un canal. Esta clase contiene varias propiedades comunes para todos los canales. Pero también se han definido algunas propiedades muy específicas de algún tipo de canal determinado. Por ejemplo, los canales de tipo ModBus TCP/IP Ethernet, tienen muchas más propiedades que el canal Siemens TCP/IP Ethernet.

Devices

En este caso se ha optado para crear una única clase para los 4 tipos de canales. Ya que las propiedades específicas de cada canal son muy pocas (entre 4 o 5) . Pero esta misma idea no se puede aplicar para crear una única clase para todos los dispositivos. Los dispositivos tienen muchas propiedades en común, pero también otras muchas que son específicas de cada dispositivo.

Se han creado 4 clases diferentes para definir las propiedades de cada tipo de dispositivo. Es decir, para el dispositivo de tipo ModBus se ha creado la clase ModBusDevice.

Todas las clases creadas para dispositivos son:

- ModBusDevice
- SimulatorDevice
- SiemensDevice
- AllenDevice

Tag
Dado que todos los tags tienen las mismas propiedades de forma común, se crea una única clase para definir todos estos campos.

TagGroup
Clase que define las propiedades que puede tener un grupo de tags.

EventLogger
Clase que contiene las 4 propiedades que tiene un evento registrado por el servidor OPC.

Response
La respuesta enviada por el servidor REST, puede tener dos formas.

Vacío: si la petición es aceptada correctamente por el servidor OPC la respuesta de este es un texto vacío que no contiene nada.

Error: si durante el procesamiento de la petición se produce algún error o excepción, el servidor en la respuesta de la petición incluye un código de error y el motivo de este.

Para recibir y tratar este tipo de respuesta, se ha creado una clase que contiene dos campos que son, el Code y Message.

ServerProperties
Clase que contiene todas las propiedades de una conexión.

UAEndpoint
Clase que contiene todas las propiedades de un OPC UA Endpoint.

ConfiguratorEvent
Esta clase es la encargada de registrar todos los eventos producidos por la propia aplicación de configuración. Esta tarea se lleva a cabo en el mismo constructor de la clase, por lo que no tiene ningún otro método declarado. Contiene 4 propiedades principales que son:

Server : es el nombre de la conexión, para identificar el servidor conectado en el momento que produjo el evento.

Source: el nombre de la sección, ventana o acción que ha lanzado este evento. Por ejemplo, si el usuario modifica el nombre de un canal, se registrará un evento, que indica el canal fue modificado por el dicho usuario. El Source en este caso será Channel Modifier.

Message: breve descripción, que indica el motivo por el cual fue producido este evento.

Date and Time: fecha y hora, de cuándo fue registrado el evento.

Esta clase no solo contiene las propiedades, sino también es la encargada de registrar los eventos en un archivo de texto.

Para registrar un evento se debe crear una instancia de la clase ConfiguratorEvent, donde en el constructor se deben enviar los parámetros Server, Source y Message. A continuación, el mismo constructor de la clase, agrega el último parámetro (Data and Time) que queda y almacena este evento en un archivo de texto llamado configurator_log.txt. De manera que cada instancia de esta clase es un evento que se va almacenando en el archivo configurator_log.txt.

Para llevar un control de cuántos eventos se han producido, se ha creado un contador. Este contador es una variable interna compartida por todas las clases del proyecto. De manera que todas las clases tienen acceso a esta variable. Cada vez que esta clase registra un evento, el contador es incrementado en uno.

Server

Clase encargada de comunicar con el servidor REST del servidor OPC, donde se han definido varios métodos que realizan peticiones de tipo GET, POST, PUT y DELETE. Cada vez que se instancia esta clase se crea un cliente REST. La configuración de este cliente REST se realizará llamando al método restClientEndpointConfig.

Este método recibe un objeto de tipo ServerProperties y con esta información actualiza el Endpoint base, es decir el tipo de protocolo (HTTP o HTTPS), la IP del servidor REST, Puerto, Usuario y la contraseña.

El constructor de la clase Server, también recibe el objeto ServerProperties y llama al método restClientEndpointConfig.

De forma que cuando se crea una instancia de la clase Server, se crea y se configura un cliente REST. Pero esta configuración del cliente REST, se puede modificar en cualquier momento llamando al método restClientEndpointConfig.

Otro de los métodos más importantes en la clase Server, es el checkConnection. Este método se utiliza simplemente para comprobar la conexión con el servidor REST. De manera que cada vez que se llama a este método, este utiliza el cliente REST para enviar una petición de tipo GET al servidor REST. Si el servidor responde correctamente a esta petición, el método checkConnection devuelve un true. En caso contrario aparte de devolver un false, este método registra un evento de tipo ConfiguratorEvent, donde el mensaje de este evento será la respuesta (error o excepción) que ha enviado el servidor REST.

La clase Server es la única de todas las clases creadas en este proyecto que se comunica directamente con el servidor REST. Por lo tanto, se han implementado más de 20 métodos (ver Anexo II) que se encargan de consultar, crear, modificar y borrar todo tipo de objetos del servidor OPC.

Hay cinco tipos de métodos (ver Tabla 5-5) creados, que son:

Tabla 5-5. Tipos de métodos implementados en la clase Server.

Tipo de método	Descripción
Create	Todos los métodos cuyos nombres empiezan con la palabra “create-------“, son para crear un objeto en el servidor OPC.

	Ejemplo : **createNewchannel (Channel channel):** este método crea un canal nuevo en el servidor OPC.
Get	Todos los métodos cuyos nombres empiezan con la palabra "Get-------", son para consultar uno o varios objetos en el servidor OPC. Ejemplo : **GetEventLoggerDefault():** devuelve los últimos 100 eventos registrados por el Servidor OPC.
Mod	Estos tipos de métodos que empiezan con la palabra "mod-------", son creados para modificar un objeto en el servidor OPC. Ejemplo: **modtag (Tag tag, string channelName, string deviceName, string tagName):** método que recibe un objeto de tipo Tag como parámetro junto con la información del canal y dispositivo. Con esta información modifica el Tag indicado en el servidor OPC.
Delete	Métodos que empiezan con la palabra "delete----" son para borrar un objeto del servidor OPC. Ejemplo: **deleteDevice (string channelName, string deviceName):** recibe un objeto de tipo Device junto con el nombre del canal al cual pertenece el dispositivo. Envía la petición de tipo DELETE para borrar el dispositivo indicado.
Otros	Hay varios métodos creados dedicados para cumplir con otras funcionalidades. Ejemplo: **checkConnection():** manda una petición de tipo Get al servidor para ver si éste está disponible. Si el servidor OPC está disponible devuelve un true.

5.4.3.2. Clases asociadas a las ventanas

Cada ventana "formulario" creado en este proyecto tiene asociada una clase. Ya sea ventana principal o secundaria. Estas clases tienen implementados varios métodos (ver Anexo IV), para hacer funcionar todos los controles en una ventana. Por ejemplo, cuando el usuario hace click sobre un botón, se produce un evento que es capturado por un método dentro de la clase.

Servers

Clase asociada a la ventana secundaria Server Connections Manager. La función principal de esta clase es mostrar, agregar, modificar y almacenar las conexiones. Todas las conexiones se almacenan en un archivo denominado servers.txt en la misma ubicación del proyecto. Los datos se guardan en un archivo ya que, de esta forma, se quedarán almacenados, aunque el usuario cierre la aplicación. Otra ventaja es que este archivo puede ser consultado por cualquier otra clase.

En esta ventana hay una tabla donde cada fila de la tabla es una conexión. El usuario puede agregar y/o borrar las filas mediante los botones "ADD NEW" y "DELETE". Una vez el usuario introduce estos datos y hace click sobre el botón "SAVE", varios métodos de la clase Servers son los responsables de convertir los datos de esta tabla en objetos de tipo ServerProperties, convertir los objetos a una única cadena de texto en formato JSON, encriptar esta cadena de texto y finalmente almacenarla en el archivo servers.txt.

Dado que una conexión contiene el nombre del usuario y la contraseña, por motivos de seguridad se ha decidido encriptar el archivo antes de almacenarlo.

Se han creado dos variables compartidas por todas las clases, Una **que registra la cantidad de conexiones** que hay almacenadas actualmente y otra **variable booleana que se pone en true** si se ha realizado algún cambio en el archivo. Estas dos variables son cruciales para que luego la clase principal (Main), pueda actualizar estos datos en la ventana principal.

Cada vez que se hace una instancia de la ventana Server Connections Manager, la clase Servers se encarga de buscar el archivo servers.txt, leer su contenido y convertir la cadena de texto en una lista de objetos tipo ServerProperties.

En el diagrama de flujo (ver Figura 5-42) se muestra el funcionamiento básico de la clase servers. Donde se puede observar, una vez se abre una nueva ventana, que la clase servers busca el archivo servers.txt. Si este archivo no existe, se procede a crearlo y el programa se queda esperando para que el usuario introduzca los datos en la tabla. En caso de que archivo si exista, se procede a desencriptar su contenido, se convierte la cadena de texto JSON en una lista de objetos de tipo ServerProperties y finalmente las propiedades de todos estos objetos se muestran en la tabla.

Finalmente, si el usuario decide guardar esta información, se realiza el proceso de convertir lista de objetos en JSON, encriptar y almacenarlo en el archivo.

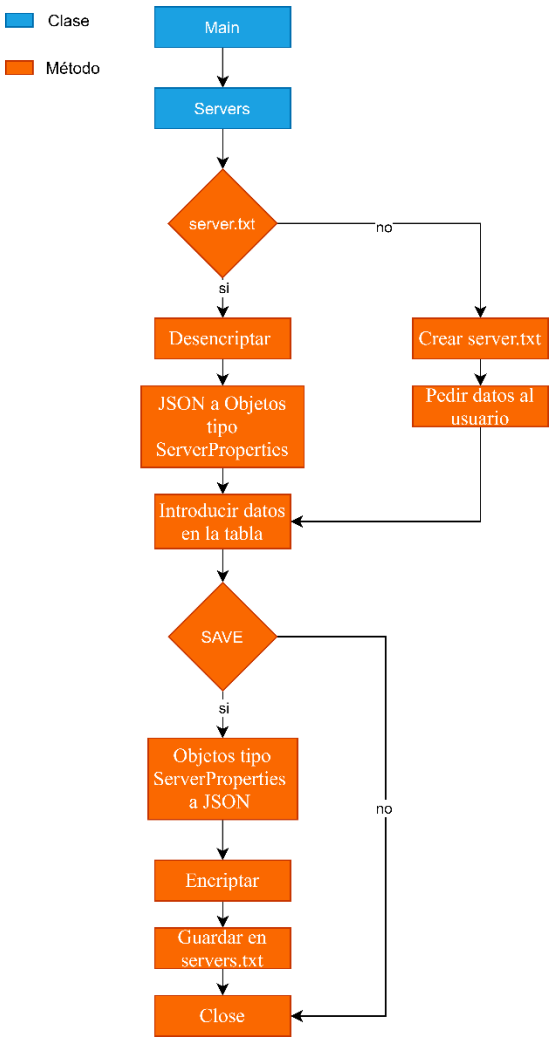


Figura 5-42. Diagrama de flujo de la clase Servers.

Main

Es la clase principal que gestionará todas las funcionalidades de la aplicación de configuración. Tiene una gran cantidad de métodos propios implementados, pero también se apoya de otras clases creadas en el proyecto. La clase main está asociada a la ventana principal, de forma que es la primera clase que se instancia y se encarga de hacer visible el formulario (ventana principal).

El constructor de la clase se encarga de cambiar el estilo “apariencia” que viene por defecto de la ventana principal. También busca el archivo servers.txt (ver **Figura 5-43**) y si este contiene algo, es decir, que existe, procede a desencriptar su contenido y convierte el JSON en una lista de objetos tipo ServerProperties y carga el nombre de todas las conexiones en una lista desplegable que hay en la sección de Connection.

Figura 5-43. Proceso de carga de los nombres de las conexiones en la clase Main.

Actualizar datos

Cada elemento de esta ventana tiene asociado varios métodos que realizan una función determinada. Pero estos métodos solo se ejecutan cuando se produce un evento por el dicho elemento. De lo contrario la clase main se queda en espera, sin realizar ninguna otra función. Si se desea realizar una función de forma cíclica "bucle", no podemos dejar ocupado el hilo principal de esta clase. Por ejemplo, si creamos un bucle que va actualizando unos datos de forma cíclica e indefinida o durante mucho tiempo, el hilo "Thread" de la clase main no puede realizar ninguna otra acción.

Una solución para este problema podría ser utilizar varios Threads en la clase principal. Pero en C# no está permitido acceder a un elemento "controlador" de un formulario desde un hilo que no sea el principal de esa clase.

Finalmente, para solucionar este problema se ha decidido utilizar temporizadores. Desde el hilo principal se puede poner en marcha un temporizador, que solamente interrumpirá al hilo principal cuando haya transcurrido el tiempo fijado para dicho temporizador.

En la clase main se han creado dos temporizadores, cada uno con su intervalo de tiempo fijado. Los dos temporizadores se ponen en marcha desde la primera vez que se ejecuta la aplicación. Se han utilizado dos temporizadores diferentes porque necesitamos dos velocidades diferentes para actualizar los datos. A continuación, se explicará la funcionalidad de cada temporizador.

En el diagrama de flujo (ver Figura 5-44) se ilustra como un temporizador se encarga de actualizar los eventos registrados por la propia aplicación de configuración. Es decir, cada vez que este temporizador interrumpe el hilo principal de la clase main, se procede a verificar si se ha producido un cambio en la cantidad de eventos registrados en el archivo configurator_log.txt. Esto es posible gracias a una variable compartida por todas las clases. Esta variable contadora se incrementa en uno cada vez que se registra un evento (una instancia de la clase ConfiguratorEvent). Una vez verificado que se ha producido un cambio en el contador y cuántos cambios se han producido, se procede a acceder al archivo configurator_log.txt, que es el que contiene todos los eventos. Se leen todos los últimos eventos que se han producido y se van añadiendo estos datos en la tabla que hay en la sección Event Log dentro de la pestaña Configurator.

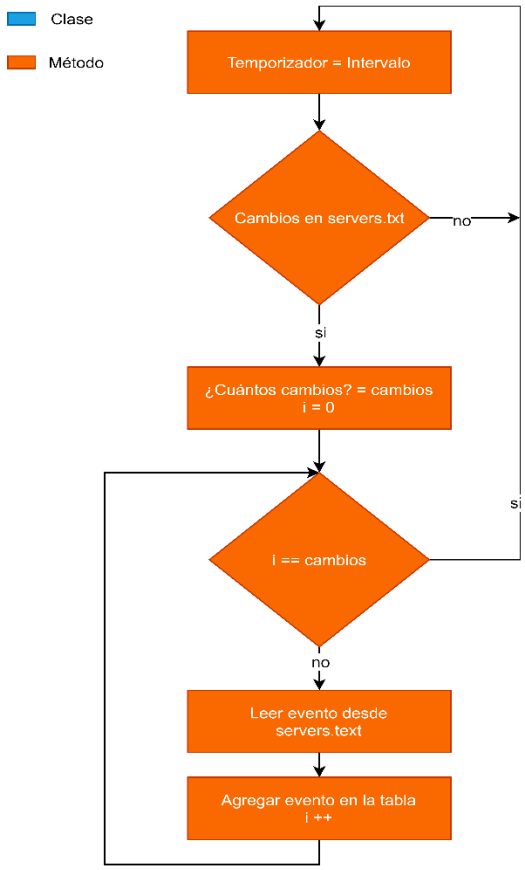


Figura 5-44. Diagrama de flujo para explicar el proceso de actualizar el contenido de la tabla Configurator.

El segundo temporizador se encarga de actualizar varios datos en la ventana principal. A este temporizador se han asignado dos misiones principales. Primero, actualizar los nombres de las conexiones en la lista desplegable que hay en la sección de conexión. Segundo, actualizar la información de la Barra de estado (mostrar la cantidad de canales, dispositivos, tags, etc.).

Actualizar los nombres de las conexiones es posible gracias a las dos variables compartidas, de las que ya hablamos en el apartado de la clase Servers (ver **Servers**). Una variable que registra la cantidad de conexiones que hay actualmente almacenadas y otra que detecta si se ha producido un cambio desde la ventana Servers Connections Manager.

Cuando algunas de estas variables (o las dos cambian de valor) y cada vez que el temporizador interrumpe se procede a leer el archivo servers.txt, descifrar el contenido y convertir la cadena de texto JSON en una lista de objetos de tipo ServerProperties. La lógica programada es la misma que se utilizó en la clase Servers.

Ahora veremos una por una todas las funcionalidades que dispone el usuario en la ventana principal, para comprender la lógica utilizada para lograr estas funciones. De nuevo esta memoria no pretende explicar el código escrito en C#, sino solamente la lógica que se ha seguido para lograr estas tareas. Para ello la explicación se apoyará de diagramas de flujos y capturas de pantallas.

Conectar con el servidor

Una vez el usuario ejecuta la aplicación de configuración o sea la ventana principal, se cargan los nombres de todas las conexiones guardadas. De esta forma el usuario ahora debe de elegir un nombre desde el desplegable y hacer click sobre el botón “CONNECT”. Esta acción lo único que hace, es hacer una prueba de comunicación con el servidor, pero esto no significa que se haya quedado conectado. Si la prueba de comunicación ha sido satisfactoria, el botón “CONNECT” cambia de estado (texto) a “DISCONNECT”. Ahora el usuario no puede modificar el nombre de la conexión desde el desplegable, hasta que no se vuelve a hacer click sobre el botón “DISCONNECT”.

Es una forma de evitar que el usuario realice alguna modificación en la conexión mientas se esté realizando alguna petición. Por ejemplo, si se están enviando peticiones de lecturas de forma cíclica, como podría ser el caso de leer el Logger del servidor OPC cada x tiempo. Mientras se está leyendo el Logger de forma cíclica y el usuario por error cambia la IP, y esta IP no existe, esto podría provocar una excepción y quizá la caída de la aplicación de configuración.

Una vez el usuario hace click sobre el botón CONNECT, se provoca una reacción en cadena de una serie de acciones para descargar toda la configuración actual del proyecto cargado al Runtime. Primero se crea una nueva instancia de la clase Server (ver **Figura 5-45**) para crear nuestro cliente REST. Al constructor de la clase Server se le pasa como parámetro el objeto de tipo ServerProperties (el objeto que contiene todos los parámetros de una conexión) seleccionado por el usuario.

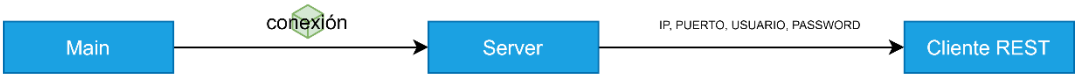


Figura 5-45. Diagrama para explicar cómo se comunica con el Servidor OPC desde la clase Main.

A continuación, se realiza una prueba de comunicación (ver **Figura 5-46**) con el servidor REST, utilizando el método checkConnection de la clase Servers. Si el servidor está disponible, se procede a descargar la configuración actual del proyecto cargado en el Runtime del servidor OPC. De lo contrario la clase Server ya se encarga de registrar un evento de tipo error indicándolo, el cual es el motivo de porqué el servidor no está disponible.

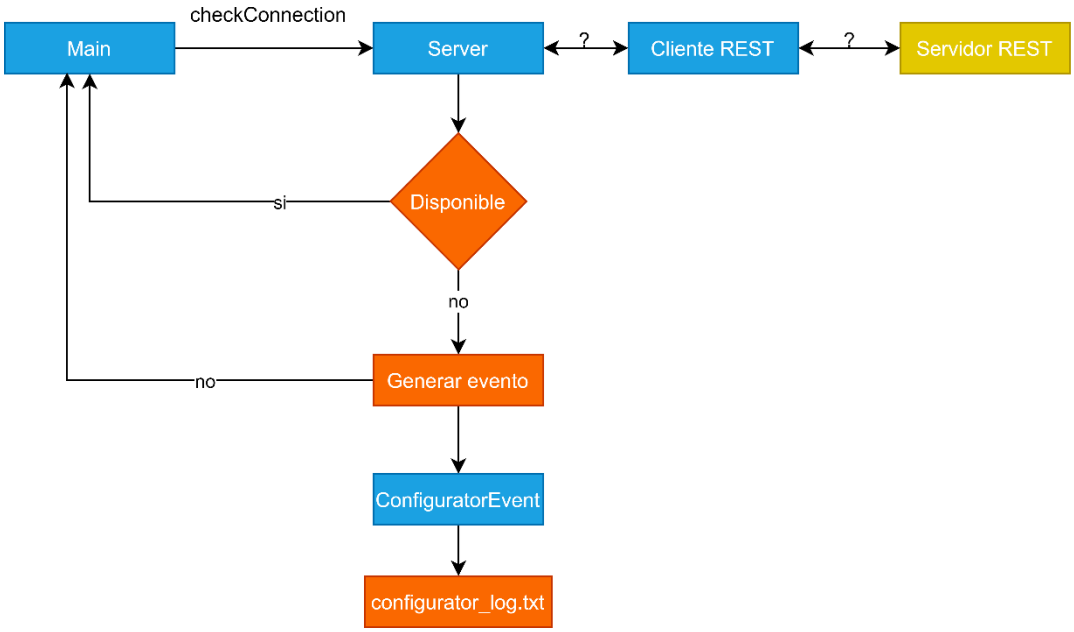


Figura 5-46. Prueba de comunicación con el Servidor REST

Si el servidor está disponible, en la clase Main se han implementado varios métodos que, utilizando la instancia de la clase Server, realizan peticiones de lectura.

Primero de todo se procede a leer todos los canales disponibles (ver **Figura 5-47**). Para ello la clase Main utiliza el método GetChannels implementado en la clase Server. Esta clase, utilizando el cliente REST, lanza la petición de tipo GET para leer todos los canales, convierte la respuesta de formato JSON a una lista de objetos tipo Channel y la devuelve a la clase Main.

La clase Main se encarga de poner el nombre de todos estos canales en una lista desplegable. Todos los parámetros del canal seleccionado son mostrados en la sección Current Configuration, en la pestaña “CHANNEL”.

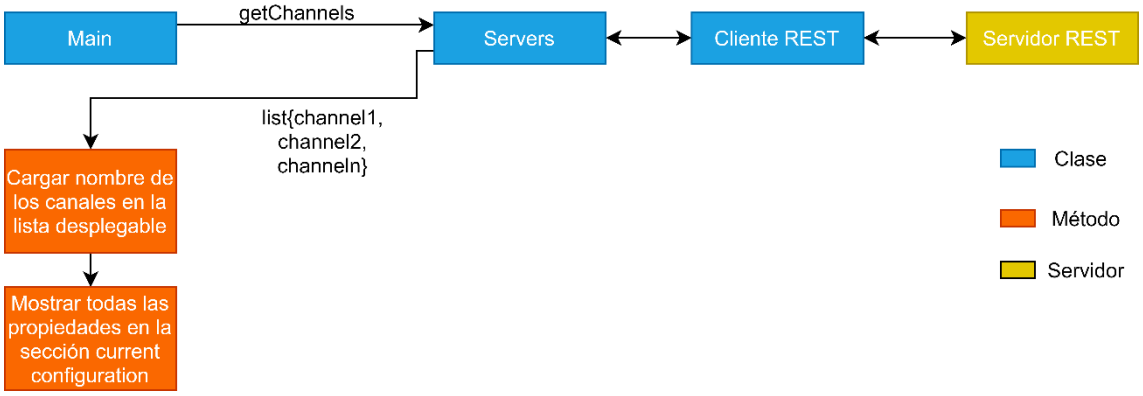


Figura 5-47. Diagrama de flujo para explicar el proceso de lectura y visualizar los canales ya creados.

A parte de mostrar todos los parámetros del canal seleccionado, el siguiente paso es volver a utilizar nuestro cliente REST para descargar todos los dispositivos pertenecientes a este canal (ver **Figura 5-48**). Pero antes de realizar este paso, es importante determinar qué tipo de canal es,

para que se pueda llamar al método correspondiente de la clase Server. Supongamos que el canal seleccionado es de tipo ModBus. Se llama al método GetModbusDevices de la clase Server pasando como parámetro el nombre del canal seleccionado. De esta manera, este método utiliza el cliente REST para lanzar la petición de tipo GET, preguntando por todos los dispositivos existentes en este canal. Este método devuelve una lista con todos los dispositivos. De igual manera que los canales, el nombre de todos estos dispositivos se muestra en una lista desplegable "Device:". Dependiendo del dispositivo que está seleccionado en la lista desplegable, se muestran todas las propiedades de este dispositivo en una tabla dentro de la pestaña "DEVICE" en la sección de Current Configuration.

De igual manera, el dispositivo seleccionado vuelve a utilizar la instancia de la clase Server y sus métodos correspondientes, para descargar y mostrar todos los tags, grupos de tags y tags pertenecientes a los grupos de tags.

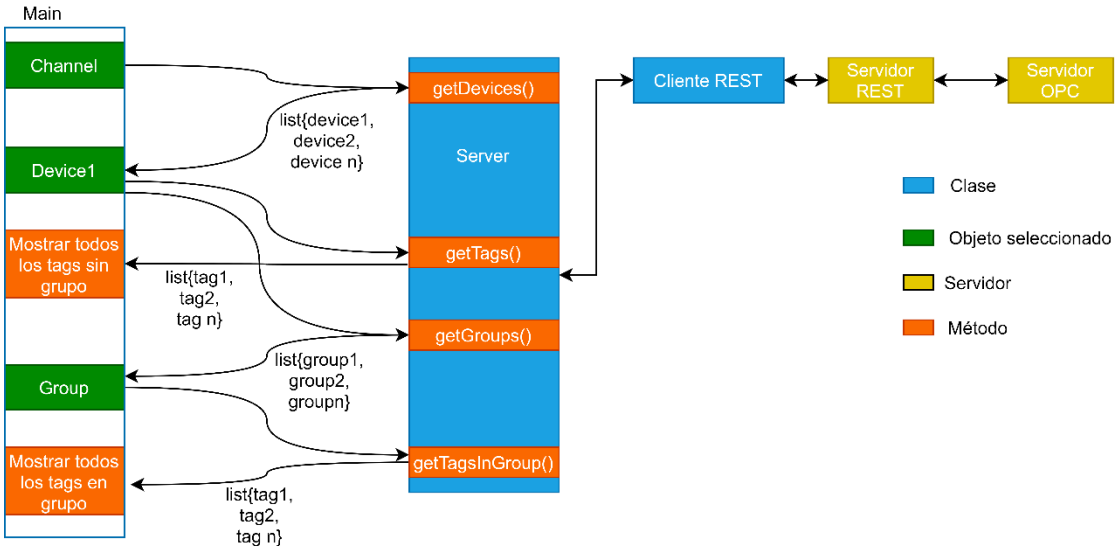


Figura 5-48. Proceso de lectura de toda la configuración actual.

Por último, esta cadena de acciones acaba cuando se crea una instancia de la clase EventLogger (ventana Server Event Logger). Y se incrusta esta ventana dentro de la pestaña Server en la sección Event Log. El nombre de esta pestaña cambia por el nombre de la conexión seleccionada.

Por ejemplo, en la figura (ver **Figura 5-49**) el usuario se ha conectado con el Servidor REST, utilizando la conexión previamente guardada, que se llama Server_1.

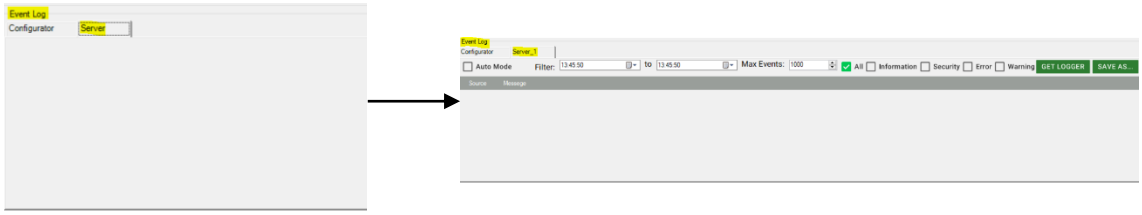


Figura 5-49. Como se agrega una ventana Server Event Logger dentro de la pestaña Server.

Todos estos pasos se realizan a partir del momento en el usuario hace click sobre el botón "CONNECT". Pero este botón lo único que hace es llamar a al botón "DOWNLOAD". Es este el

botón que realmente tiene implementados todos los métodos para descargar toda la información. Por lo tanto, una vez conectado si el usuario quiere volver a leer toda la configuración ,o sea volver a descargar, puede hacer click sobre el botón “DOWNLOAD”.

Otra forma de actualizar la información es provocando un cambio en la selección de un elemento. Por ejemplo, si se selecciona otro canal, esto provoca un cambio en la lista desplegable, que provocaría otra vez la cadena de acciones vistas anteriormente.

Modificar configuración actual

Una vez descargada y mostrada la configuración actual, el usuario tiene la posibilidad de modificar esta configuración en el Servidor OPC. Para que el usuario pueda modificar, se deben cumplir con dos requisitos fundamentales. Uno, que el usuario esté conectado a un servidor y segundo, solo se puede modificar la configuración que se está mostrando en la sección de configuración actual.

Todas las pestañas de la sección de configuración actual disponen de un botón denominado “MODIFY”. Todas las propiedades mostradas en esta pestaña están bloqueadas (no permite modificar) hasta que el usuario hace click sobre el botón “MODIFY”. Supongamos que el usuario quiere cambiar el nombre y la descripción de un canal llamado channel1. Para ello el usuario selecciona el canal Channel1 desde la lista desplegable de todos los canales cargados. A continuación, la aplicación muestra todas las propiedades de este canal en la pestaña “CHANNEL”. Desde esta pestaña el usuario hace click sobre el botón “MODIFY”. De forma que ahora puede modificar una o varias de las propiedades. Y hace click sobre el botón “UPLOAD”. Este botón tiene asociado un método que se encarga de crear un objeto nuevo de tipo Channel, lee todas las propiedades modificadas por el usuario, del canal desde la pestaña “CHANNEL” y guarda todas esas propiedades en el objeto creado.

Este objeto junto con el nombre del canal actual, son enviados al método modifyChannel de la clase Server (ver Figura 5-50). De esta forma este método sabe, a que canal y qué propiedades hay que modificar. Este método se encarga convertir y enviar esta información al Servidor OPC. Una vez lanzada la petición para modificar el canal, si la respuesta del servidor es buena, este método devuelve un true a la clase Main, de manera que este genera un evento de tipo ConfiguratorEvent. Este evento es una advertencia que indica al usuario que el canal fue modificado. Si durante la petición de modificación el Servidor REST ha devuelto alguna excepción o error, se generará otro evento de tipo ConfiguratorEvent que incluye como mensaje el motivo por el cual el canal no ha sido modificado. Y el método modifyChannel devolverá un false a la clase Main.

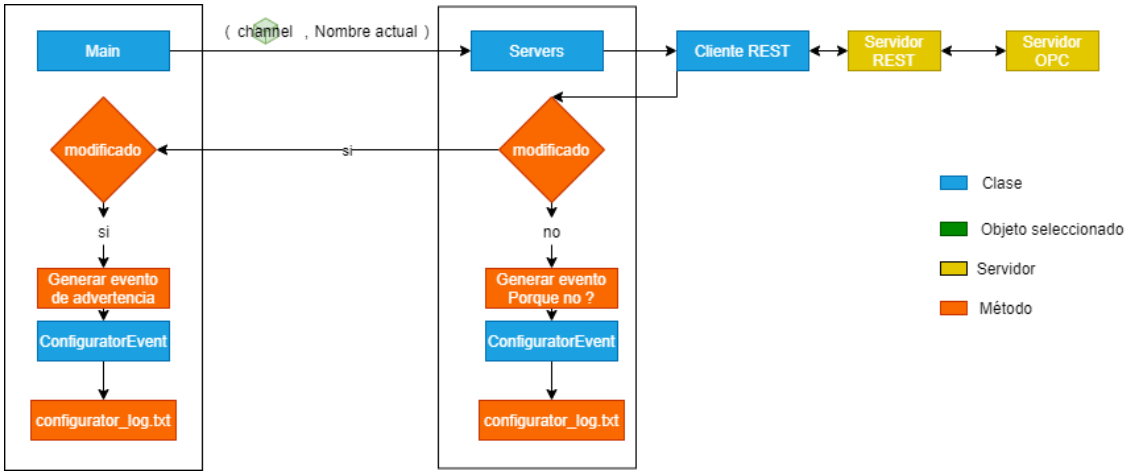


Figura 5-50. Diagrama de flujo que explica cómo se modifica un canal.

Esta misma idea se aplica para modificar los dispositivos y los tags. La única diferencia es que, para modificar un dispositivo, se enviará un objeto que contiene todas las propiedades modificadas junto con el nombre del canal y del dispositivo actual. De forma que el método correspondiente en la clase Server, sepa qué parámetros, en qué canal y dispositivo hay que modificar.

Para modificar los tags, una vez el usuario modifica todos los tags necesarios desde la pestaña “TAGS” o “GROUP TAGS” y a continuación, hace click sobre el botón “UPLOAD”, se vuelven a leer todos los tags y se almacenan en una nueva lista de objetos tipo Tag. Acto seguido, mediante un bucle se envían uno a uno estos tags a la clase Server llamando al método correspondiente.

Los métodos implementados para esta tarea de la clase server se encargan de enviar las peticiones de modificación al Servidor OPC. En caso de que haya algún problema “error o excepción” para modificar un tag, la clase Server genera los eventos necesarios para informar al usuario, indicando el motivo por el cual el tag no se ha modificado.

En la siguiente captura (ver **Figura 5-51**), se ilustra el comportamiento programado para modificar la configuración actual. Donde el usuario modifica las direcciones de algunos tags que contiene el dispositivo Device_1. Pero por error el usuario ha introducido mal las direcciones de los tags Tag0,Tag1,Tag3 y el Tag5. Una vez lanzada la petición para modificar los tags desde la clase Server, el servidor OPC responde indicando qué tags contienen error de sintaxis en las direcciones. Estos errores se convierten en eventos de tipo ConfiguratorEvent, que a posteriori serán mostrados en la sección de Event Log en la ventana principal.

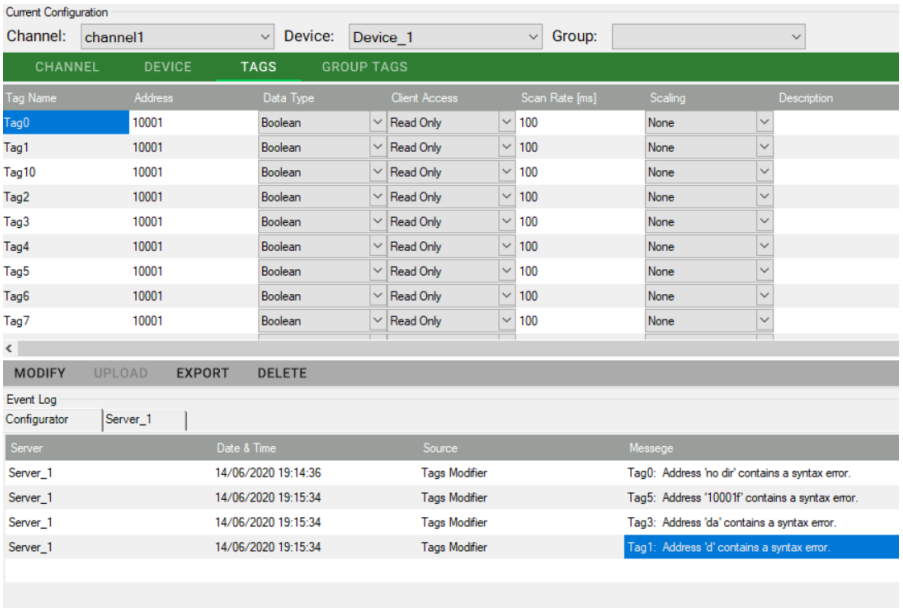


Figura 5-51. Captura de la pantalla que muestra los eventos que registra la clase Server, cuando no se ha podido modificar un tag.

Borrar la configuración actual

Otra de las funcionalidades a implementar es permitir al usuario borrar la configuración actual. Ya sea un canal, dispositivo o un tag. Para ello a la API REST del servidor OPC en la URI hay que enviar el nombre del objeto a borrar y la petición de tipo DELETE. En la clase Server se han incluido todos los métodos necesarios para realizar esta tarea. Estos métodos reciben como parámetro toda la información necesaria para acabar de construir la URI del objeto a borrar.

Cada pestaña de la sección Current Configuration tiene un botón denominado “DELETE.” Si estando en la pestaña “CHANNEL”, el usuario hace click sobre este botón, por motivos de seguridad la aplicación mostrará una ventana de confirmación (ver **Figura 5-52**) para asegurar que el usuario realmente quiere borrar este canal.

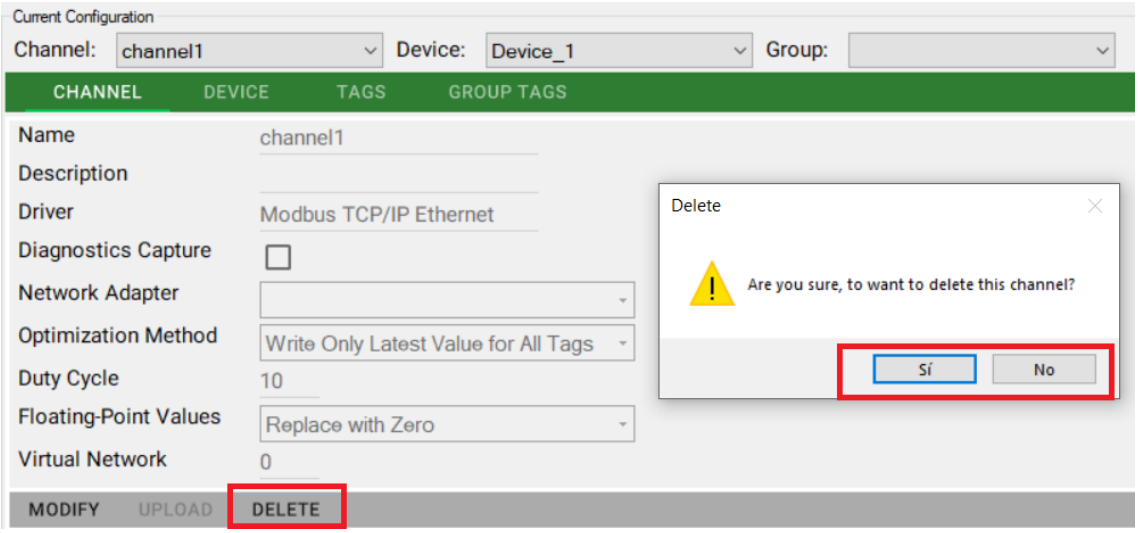


Figura 5-52. Ventana de confirmación antes de borrar un objeto.

Si el usuario confirma que quiere borrar este canal se procede a llamar al método deleteChannel de la clase Server. Este método recibe el nombre del canal como parámetro. Lanza la petición de tipo DELETE utilizando el cliente REST de la clase Server (ver **Figura 5-53**).

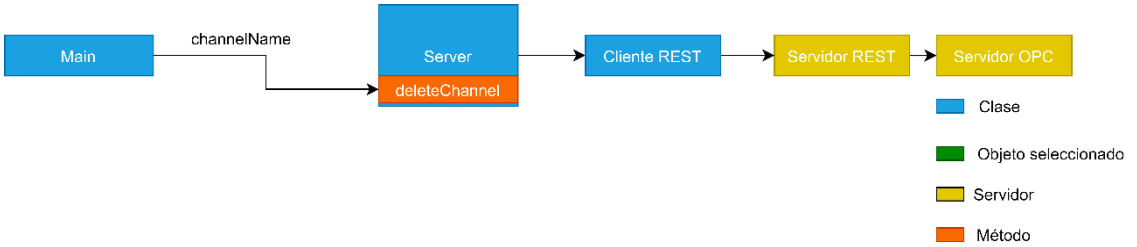


Figura 5-53. Diagrama de flujo que explica el proceso para borrar un canal.

La misma idea se aplica para borrar todos los objetos restantes. Es decir, los dispositivos, tags y tags dentro de un grupo. A la hora de borrar los tags no es posible borrar todos los tags a la vez. De forma que, el usuario para borrar un tag, debe seleccionarlo y a posteriori hacer click sobre el botón “DELETE”.

Cada vez que se borra un elemento, por motivos de seguridad se genera un evento en el Logger de la aplicación de configuración, que indica qué elemento y por qué usuario ha sido borrado (ver **Figura 5-54**).

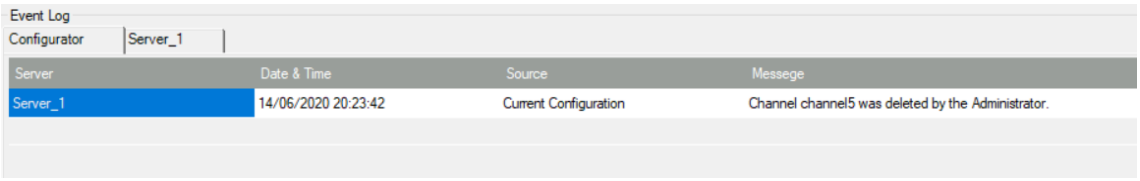


Figura 5-54. Captura de la pestaña Configurator, donde se observa el evento que se registra por la clase Main después de borrar un objeto.

Exportar tags

Esta funcionalidad permite al usuario exportar todos los tags que se están mostrando actualmente, ya sea en la pestaña “TAGS” o “GROUP TAGS”. Estos tags se exportan un archivo de texto.

Para ello en la pestaña “TAG” y “GROUP TAGS”, hay un botón denominado “EXPORT”. Cada vez que el usuario hace click sobre este botón se abre una nueva ventana (ver **Figura 5-55**), para que el usuario ponga un nombre del archivo y elija la ubicación donde se va a generar.

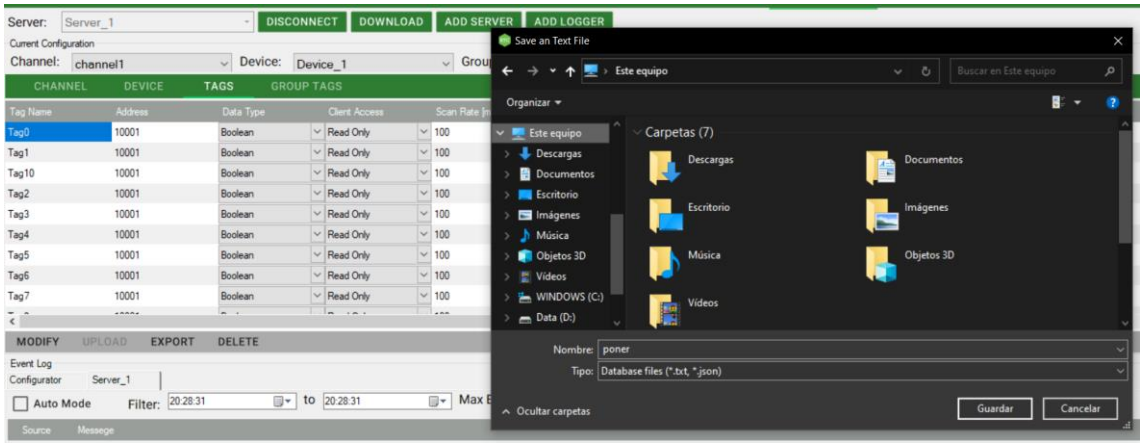


Figura 5-55. Captura de la pantalla a la hora de exportar los tags.

El proceso de exportación de los tags comienza (ver **Figura 5-56**), cuando el usuario hace click en el botón EXPORT. Seguidamente, el método asociado a este botón verifica si hay tags disponibles para exportar. En caso de que no existan los tags, no se tomará ninguna acción. Pero si hay tags disponibles para exportar, se convierte la lista de objetos de tipo TAG en una cadena de texto con formato JSON (ver **Figura 5-57**). Se abre una ventana que pide al usuario la ubicación dónde se va a guardar el archivo.

Tras determinar la ruta del archivo, se procede a crear un archivo de texto cuyo contenido es la cadena de texto que se ha creado previamente. Por motivos de seguridad se registrará un evento de tipo ConfiguratorEvent para dejar constancia desde qué servidor, a qué hora, quién ha exportado los tags y cuántos tags se han exportado.

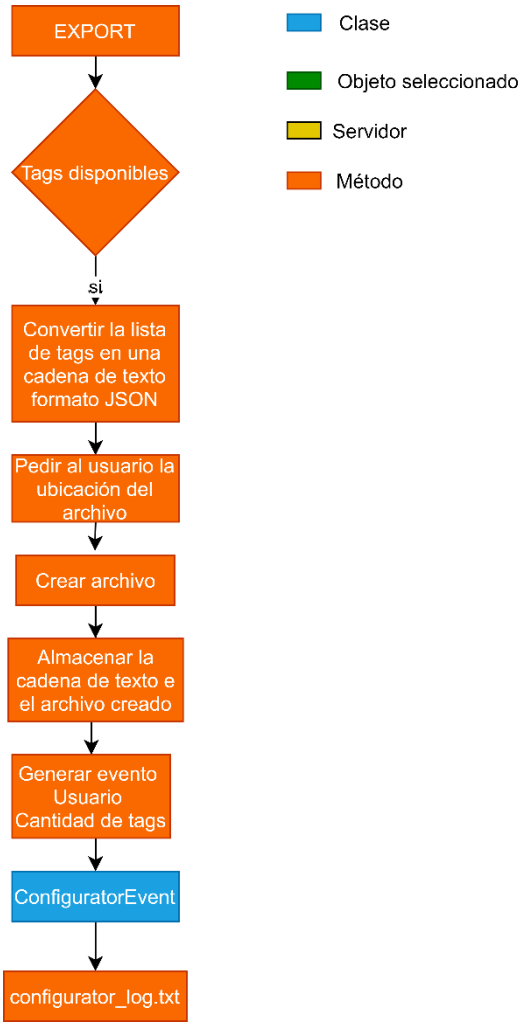


Figura 5-56. Diagrama de flujo que muestra el proceso de exportación de los tags.

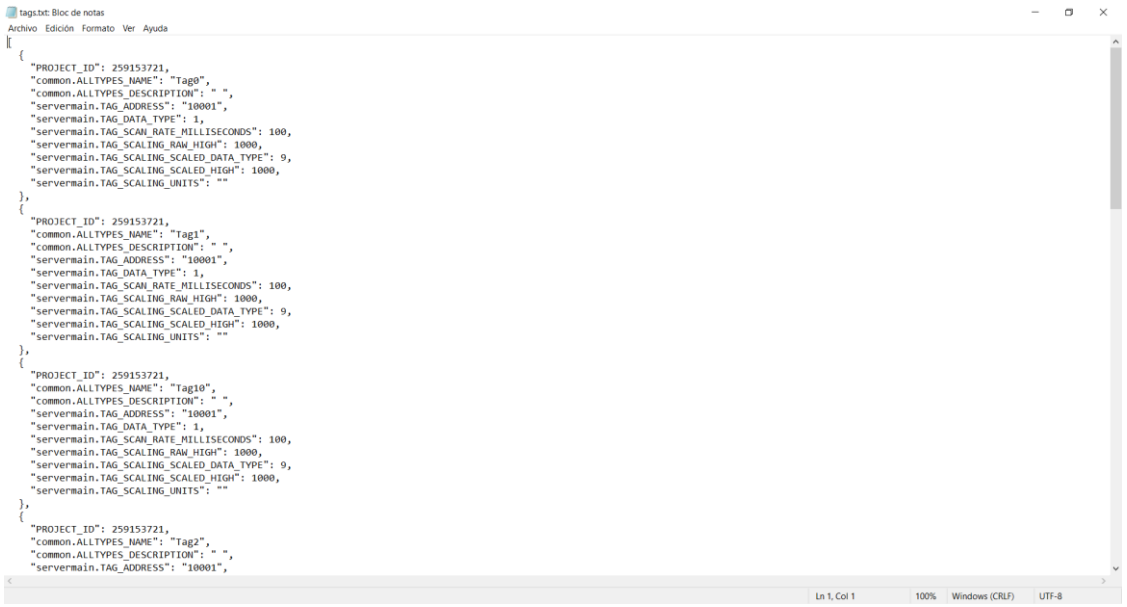


Figura 5-57. Captura de la pantalla que muestra el archivo que contiene los tags exportados.

Consultar varios Event Logger

Uno de los requisitos y retos planteados para diseñar la aplicación de configuración es, desde la propia aplicación de configuración, permitir al usuario leer varios Event Logger a la vez. De forma que el usuario pueda consultar el Event Logger de un mismo o varios servidores simultáneamente.

Este es el motivo principal por el cual se ha creado una ventana secundaria denominada Server Event Logger. La idea es que, en vez de crear varias pestañas ya prefijadas, cada una con su tabla para mostrar el Event Logger, crear una única ventana secundaria con sus métodos y clases. Que pasando los parámetros adecuados muestre el Event Logger de un servidor determinado. Entonces, incrustar esta ventana secundaria en la ventana principal, haciendo tantas instancias como sean necesarias.

Otra de las ventajas de crear una ventana secundaria (ver Figura 5-58) es, mejorar el tiempo de carga de los Eventos en las tablas. Es decir, si consultamos el Event Logger de varios servidores desde la ventana principal (la clase Main), estamos obligados a introducir los datos recibidos por los servidores de forma de serie, es decir, solo un Event Logger a la vez. Podríamos utilizar varios Threads, para que las consultas sean de forma paralela, pero para introducir estos datos en la tabla no podemos acceder a un control “tabla” desde un Thread que no sea el principal. Por lo tanto, si utilizamos ventanas secundarias, cada una con su propio Thread, mejorará el rendimiento de la aplicación en general.

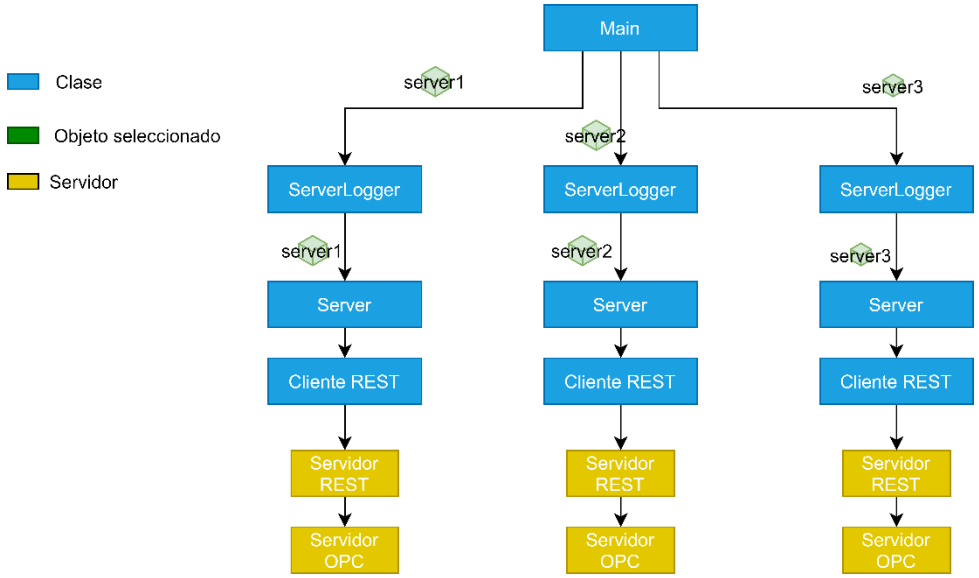


Figura 5-58. Forma de crear varias ventanas de tipo Server Event Logger.

El funcionamiento de la ventana secundaria se explicará más adelante en un apartado dedicado especialmente para ServerLogger. Por ahora se enfocará solo en cómo se agregan estas ventanas en la ventana principal.

Para agregar un Server Event Logger, se ha creado un botón “ADD LOGGER” (ver **Figura 5-59**) en la sección de conexión en la ventana principal. Se ha programado de forma que, cuando el usuario hace click sobre el botón aparece un menú en forma de lista con los nombres de todas las conexiones guardadas. Una vez el usuario selecciona un nombre. Se crea una nueva pestaña en la sección de Event Log. El nombre de la pestaña será el mismo que la conexión seleccionada. Seguidamente, se crea una nueva instancia de la clase ServerLogger, que es la clase asociada al formulario “ventana” Server Event Logger. El constructor de esta clase recibe como parámetro, el objeto de tipo ServerProperties. Finalmente, se agrega y se muestra esta nueva ventana dentro de la nueva pestaña creada.

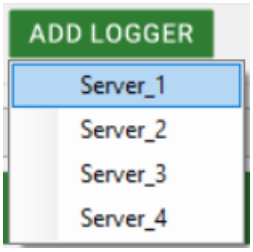


Figura 5-59. Botón "ADD LOGGER" con la lista de los nombres de todas las conexiones guardadas.

Nueva configuración

Uno de los requerimientos de la aplicación de configuración es permitir al usuario, crear nueva configuración. Es decir, crear un nuevo canal, dispositivo, tags o grupo de tags. Para ello fue creada la sección New Configuration, que está dividida entre tres pestañas que son “NEW CHANNEL, NEW DEVICE y NEW TAG”.

Crear nuevo canal

Desde la pestaña “NEW CHANNEL”, el usuario puede introducir todas las propiedades que tiene un canal. Para ello se han creado varios controles (cuadros de textos, desplegables, check box, etc.). La cantidad de estos controles es directamente proporcional a la cantidad de propiedades que puede tener un canal. Es decir, si el canal es de tipo ModBus se mostrarán muchas más propiedades que un canal de tipo Simulator. Esta capacidad de adaptación de los controles es posible gracias al desplegable “Driver”(ver **Figura 5-60**). Esta lista desplegable por defecto incluye los cuatro tipos de posibles drivers que soporta el servidor OPC. Está programada de

forma que, cuando el usuario elige un tipo de driver en concreto, se procede a dar valores por defecto a algunas de las propiedades restantes que aún el usuario no ha introducido. De esta forma las únicas propiedades que el usuario está obligado a introducir son el nombre del canal a crear y elegir el tipo de driver a utilizar. Por supuesto el usuario puede modificar los valores que se han dado por defecto.

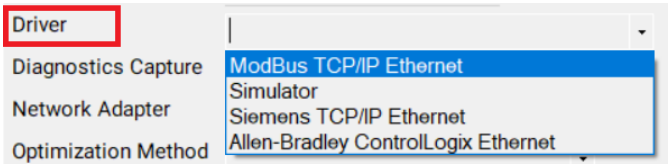


Figura 5-60. Lista desplegable que muestra los drivers soportados

Una vez introducidos los datos, el usuario hace click sobre el botón “CREATE”. El método asociado a este botón, primero de todo, comprueba que el usuario está conectado a un servidor y luego si se han introducido los parámetros obligatorios (nombre del canal y tipo de driver). En ambas comprobaciones si estos parámetros no están introducidos, se mostrará una ventana de tipo popup qué indica que parámetro se debe de introducir (ver Figura 5-61).

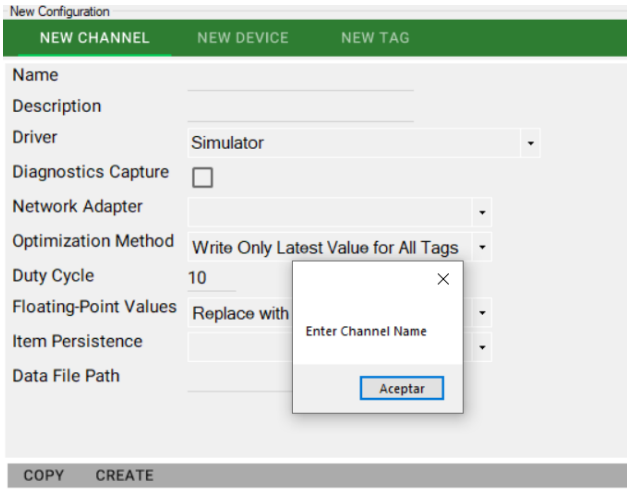


Figura 5-61. Captura de pantalla que muestra la ventana popup, en caso de que el usuario no haya puesto el nombre del canal a crear.

Una vez comprobado los parámetros, se crea un objeto de tipo Channel, se leen todas las propiedades y se almacenan en este objeto. Seguidamente, se llama al método createNewchannel de la clase Server y este método recibe como parámetro el objeto Channel con los datos rellenados. El método ya se encarga de comunicar con el servidor OPC para enviar estos datos. También se encarga de recibir la respuesta del servidor. Si el canal se ha creado correctamente devuelve un true, de esta manera la clase Main ya se encargará de registrar un evento “advertencia” que indica que el canal se ha creado correctamente (ver Figura 5-62). En caso de que haya habido un problema “excepción” el método createNewchannel devuelve un false, también analiza la respuesta del servidor y genera los eventos necesarios, indicando el motivo por el cual el canal no fue creado.

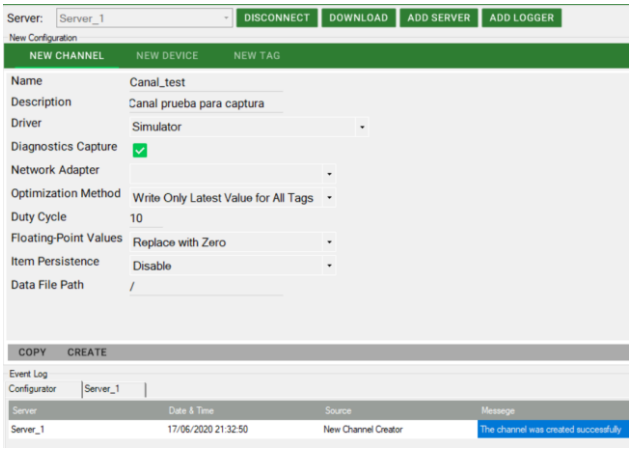


Figura 5-62. Captura donde se puede observar el evento que se registra, cuando un canal se ha creado correctamente.

Después de crear el canal, se llama al método asociado al botón "DOWLOAD" que se encarga de descargar de nuevo la configuración actual del servidor. De manera que el usuario en la aplicación de configuración puede ver el canal que acaba de crear.

Otro de los requisitos para programar la aplicación es permitir al usuario crear un canal o dispositivo a partir de un canal ya existente "copiar un objeto" (ver Figura 5-63). Para cumplir con este requisito se ha creado un botón llamado "COPY". Este botón copia y muestra las propiedades de un canal que ya está creado y descargado en la aplicación de configuración. El único requisito para utilizar esta opción es que tiene que haber un canal ya descargado y mostrándose en la sección de Current Configuration.

Obviamente no puede haber dos canales con el mismo nombre. Para solucionar esto a la hora de copiar un canal se han implementado métodos que se encargan de analizar el nombre del canal a copiar y agregar una numeración ascendente al final del nombre. Un ejemplo para comprender estos métodos sería el siguiente.

Supongamos que estamos copiando un canal llamado "Channel_1". La lógica programada se encarga de detectar de que este canal tiene un "_1" al final del nombre. Por lo tanto, el nombre del canal copiado será "Channel_2". En caso de que el nombre del canal a copiar no tenga la terminación de tipo "_x", se coloca "_1" al final del nombre del canal copiado.

Una vez copiado el canal, el usuario puede modificar los parámetros, antes de hacer click sobre "CREATE" para crear el canal.

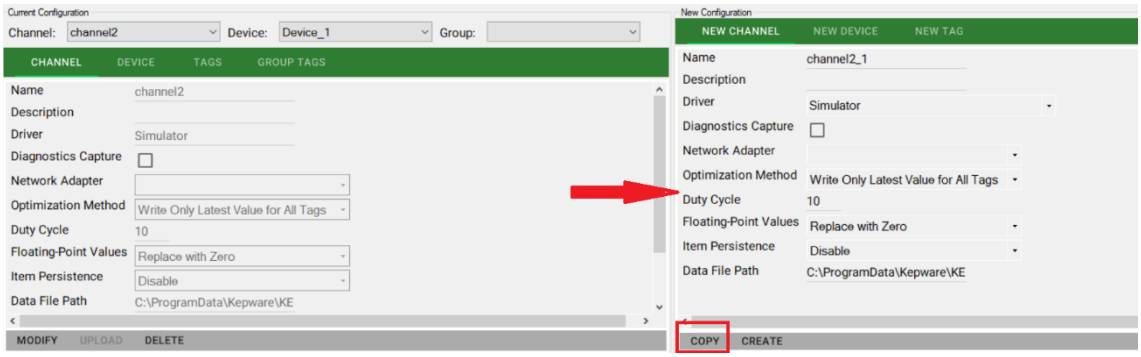


Figura 5-63. Captura que muestra como un canal es copiado después de hacer click sobre el botón "COPY".

Crear nuevo dispositivo

Tras crear el canal, el siguiente paso suele ser crear un nuevo dispositivo en el canal creado. Para ello el usuario se dirige a la siguiente pestaña “NEW DEVICE”. Aquí el usuario, mediante una lista de selección, puede filtrar los canales ya creados. Al seleccionar el canal automáticamente, dependiendo del tipo de canal que sea (tipo de driver), en una tabla que hay en esta pestaña se cargan todas las propiedades que puede tener un dispositivo. Esta tabla se adapta dependiendo de la cantidad de propiedades que tenga este tipo de dispositivo. A parte de cargar las propiedades también se dan algunos valores por defecto a algunas de las propiedades, que son obligatorias para crear un dispositivo.

Finalmente, cuando el usuario hace click sobre el botón “CREATE” el método asociado a este botón se encarga de comprobar qué tipo de dispositivo es, mirando la propiedad llamada driver. Seguidamente, se crea una instancia de un objeto, dependiendo del tipo de dispositivo. Aquí puede haber cuatro posibilidades. Supongamos que el dispositivo a crear es de tipo Modbus. Entonces el objeto a instanciar seria ModBusDevice.

Tras crear el objeto, mediante un bucle se procede a leer y guardar los datos de la tabla en el objeto creado. A su vez se comprueba que todas las propiedades obligatorias están introducidas en la tabla. Para crear este dispositivo se llama al método creatNewMdbDevice de la clase Server. Este método recibe como parámetro el objeto de tipo ModBusDevice que hemos creado, que ya contiene toda la información necesaria para crear un dispositivo nuevo. Como todos los métodos de tipo create que se han visto hasta ahora, este también se encarga de enviar la petición al servidor OPC, recibir la respuesta y analizarla, para crear los eventos necesarios, en caso de que la petición no se haya realizado de forma correcta o se ha generado alguna excepción “error”.

Crear nuevos tags

En la pestaña “NEW TAG” se ha creado una lista de selección y 3 desplegables (ver **Figura 5-64**). En la lista de selección se han agregado 4 elementos de texto que indican los 4 tipos de drivers que hay. Cuando el usuario selecciona un driver, de la misma forma que ya vimos en el apartado de descargar la configuración actual, se proceden a descargar todos los canales existentes en el servidor OPC. Los nombres de estos canales descargados se cargan en el primer desplegable denominado “Channel:”. La misma idea se ha usado para programar este desplegable, de forma que cuando el usuario elija un canal, se consultan al Servidor OPC los dispositivos pertenecientes a ese canal. Los nombres de estos dispositivos se cargan en segunda lista desplegable “Device:”.

El dispositivo seleccionado provoca la llamada a los métodos correspondientes para descargar y cargar los grupos de tags en el tercer desplegable “Group:”.

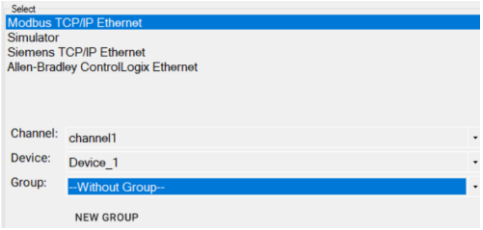


Figura 5-64. captura que muestra la lista de selección y los tres desplegables.

En caso de que el dispositivo no tenga ningún grupo, el texto a mostrar en el desplegable será “--WithoutGroup--”. A su vez en este caso también se hace visible el botón “NEW GROUP”, para que el usuario pueda crear un nuevo grupo.

Para crear los tags, se ha creado una tabla dentro de esta pestaña. En esta tabla cada fila corresponde a un tag. Es decir, cada columna de esa fila es una propiedad del tag. Para agregar un tag el usuario debe hacer click sobre “ADD”. Este botón se ha programado para que introduzca una nueva fila (un nuevo tag) en la tabla, dando unos valores por defecto a cada propiedad.

Otro botón programado es el “REMOVE”, que sirve para borrar de la tabla la fila seleccionada. El botón “REMOVE ALL” limpia toda la tabla.

Aunque el botón “ADD” da por defecto un valor a cada propiedad contenida en la fila, el usuario puede interactuar con esta tabla para modificar todas las propiedades mostradas.

También se pueden importar tags desde un fichero de texto. El funcionamiento y la lógica programada para esta funcionalidad se explicará en el siguiente apartado [Importar Tags].

Una vez agregados todos los tags necesarios en la tabla, el usuario, para crear los tags, debe hacer click sobre el botón “CREATE”. Este botón se ha programado de forma que comienza leyendo todas las filas de la tabla y guarda esta información en una lista de objetos de tipo Tag.

De forma que ahora tenemos una lista de tags con todas las propiedades configuradas por el usuario. Tras crear la lista de objetos estos son enviados a la clase Server, llamando al método creatNewTags. Este método recibe la lista de objetos de tipo Tag, los nombres del canal, dispositivo y grupo de tags seleccionados.

Aquí puede haber dos posibilidades que debe evaluar este método. El nombre del grupo de tags que se pasa a ese método puede ser un nombre del grupo real o la palabra clave “--WithoutGroup--”.

El método está programado para que detecte esta situación y construya la URI adecuada. Ya que la URI depende de si los tags se van a crear dentro de un grupo o no. Recordemos que la URI es la dirección de un objeto determinado dentro del proyecto cargado al Runtime del servidor OPC.

Una vez construida la URI envía estos tags al servidor OPC. Si todos los tags se han creado correctamente (ver Figura 5-65) devuelve un true. Pero si no se crea uno o varios de los tags (ver

Figura 5-66) el método analiza la respuesta del servidor y genera los eventos necesarios para indicar cuál es el problema.

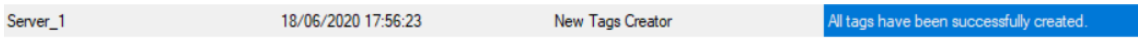


Figura 5-65. Captura que muestra el evento que registra la clase Main si todos los tags se han creado correctamente.

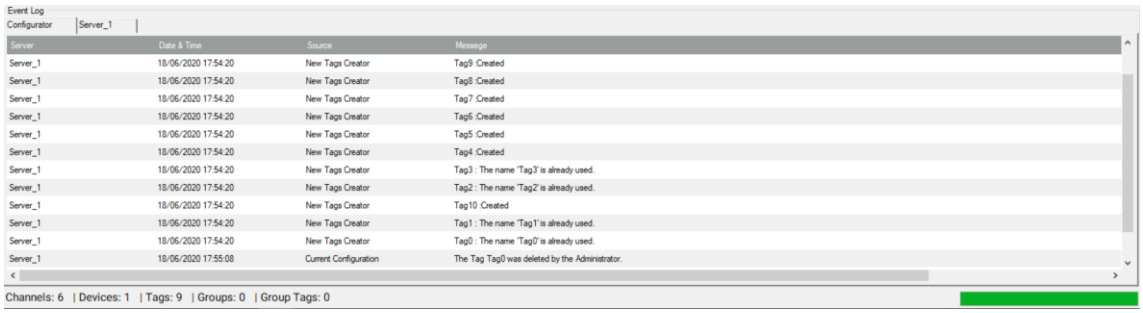


Figura 5-66. Captura que muestra los eventos que registra la clase Server, cuando algunos de los tags no se han creado correctamente.

Importar Tags

Para dotar de esta funcionalidad a la aplicación de configuración, se ha utilizado el proceso inverso de la importación de tags. En la sección de New Configuration, dentro de la pestaña "NEW TAG", se ha creado un botón denominado "IMPORT". Este botón se ha programado de forma que, si el usuario hace click sobre este botón, se abre una nueva ventana donde el usuario deberá seleccionar el archivo a importar.

De igual manera que el archivo exportado desde la aplicación de configuración los tags guardados tienen formato de texto JSON, el archivo importado también deberá tener el mismo formato de texto.

Una vez seleccionado el archivo, varios métodos se ponen en marcha para detectar si realmente existe o no este archivo. Si existe, se comprueba si el contenido de este es válido o no. En caso de que la cadena de texto no tenga el formato correcto, o sea, que contenga algún error de sintaxis, se registra un evento en la aplicación de configuración (pestaña configurator), indicando cuál es el motivo por el cual no se han podido importar los tags.

Tras verificar que el archivo es válido, se procede a convertir la cadena de texto JSON, en una lista de objetos de tipo Tag. Finalmente, toda esta información es introducida en la tabla que hay dentro de la pestaña "NEW TAG". Después de mostrar los tags en la tabla, el usuario puede modificar esta información, ya sea cambiando alguna propiedad de un tag o borrándolo de la lista.

Barra de estado

En esta sección se han implementado dos funcionalidades fundamentales que proporcionan información relevante al usuario de una forma rápida y práctica. Primero, mostrar la cantidad de canales, dispositivos en el canal seleccionado, tags dentro del dispositivo, grupos y cantidad de tags dentro del grupo seleccionado.

Para ello en esta sección se han creado varias etiquetas. El texto de estas etiquetas contiene toda esta información a mostrar. Si el usuario está conectado con un servidor y ha descargado la configuración actual, se aprovechan todas estas listas de objetos, para contar la cantidad de objetos de cada una y, así, mostrar en textos estas etiquetas creadas en la barra de estado.

La segunda función implementada, es una barra de progreso que sirve para indicar al usuario si un proceso está en marcha, si ha acabado o no y cuánto queda para acabar el proceso que está en marcha.

Para explicar la lógica aplicada para programar esta barra de progreso, a continuación, se apoyará de un ejemplo.

Supongamos que el servidor OPC actualmente tiene 12 canales. El usuario descarga todos estos canales, lanzando la petición adecuada y cuando ya tenemos una lista de objetos con los 12 canales, para cargar los nombres de estos 12 canales, en el desplegable se utiliza un bucle, que con cada iteración va accediendo a cada uno de los objetos de la lista y lee la propiedad que contiene el nombre del canal y agrega este nombre en la lista desplegable.

Aquí es donde entra en acción la barra de progreso para indicar al usuario que hay un proceso en marcha que está cargando los nombres de todos estos canales en la lista desplegable.

La implementación es sencilla ya que lo único que hay que hacer es, antes de empezar el bucle, se configura la barra de progreso para que se adapte a este bucle en concreto. El mínimo de la barra será 0, el máximo es 12 y el incremento será de unidad. De forma que cada iteración del bucle aumenta el valor de la barra en unidad.

Este solo era un ejemplo, ya que esta funcionalidad se ha aplicado en varios procesos que tienen lugar durante la ejecución de la aplicación.

Barra de menús

File, es un menú que por ahora solo tiene un único submenú “Exit”. Para implementar este botón se llama al método Close() de la ventana principal. De esta forma se consigue cerrar la aplicación.

Por ahora solo tiene el botón “Exit” ya que en un futuro se implementarán más funcionalidades.

El grupo **View** tiene 4 submenús, que sirven para mostrar o esconder las 4 secciones de la ventana principal. Para esconder o mostrar una sección en concreto se juega con su visibilidad. Esto es posible gracias una propiedad (Visible) nativa de los controles en .NET. Para una mayor comodidad de usuario se ha decidido guardar la configuración de las View de forma permanente. Es decir, si el usuario ha decidido esconder una sección y cierra la aplicación. Cuando se vuelva a abrir la aplicación dicha sección debería seguir estando escondida.

Para ello se han creado 4 variables internas de configuración de usuario. Estas variables se guardan de forma permanente y se pueden acceder para consultar o modificar desde cualquier clase. Por lo tanto, antes de cerrar la aplicación se guarda el estado de cada una de las variables que indican la visibilidad de cada sección. Y el constructor de la clase Main, vuelve a leer las 4 variables internas, para decidir mostrar o esconder las secciones.

El grupo de menús **Connection**, contiene cuatro submenús que son, “Connection”, “Edit Server Connections”, “MultiLogger” y “OPC UA Endpoints”. El objetivo de este grupo de menús es dar

acceso al usuario a algunas de las ventanas secundarias. Por ejemplo, cada vez que el usuario hace click sobre el menú MultiLogger, se ha implementado un método asociado a ese menú para que este se encargue de crear una nueva instancia de la clase MultiLogger, que es la clase asociada a la ventana Muti Event Logger. Creada la instancia, también el mismo método se encarga de mostrar por la pantalla esta ventana secundaria, pasando los parámetros necesarios a su constructor.

La misma idea se aplica para las otras ventanas secundarias, que se pueden acceder desde este grupo de menús.

Finalmente, el último menú **Help** también abre una nueva ventana secundaria de tipo About, utilizando la misma idea anterior.

ServerLogger

Clase asociada a la ventana secundaria Server Event Logger. El objetivo principal de esta clase es comunicar con el Servidor OPC y leer su Event Logger. En el apartado de diseño vimos cual es la funcionalidad de esta ventana secundaria y también se ha hablado de donde se emplea esta ventana. Ahora nos centraremos en cómo funciona, qué clases y métodos están implicados (ver Figura 5-67).

El constructor de esta clase recibe como parámetro el objeto de tipo ServerProperties. De manera que, utilizando las propiedades de una conexión, puede crear una instancia de la clase Server. De forma que la clase ServerLogger puede comunicarse con el servidor OPC utilizando el cliente REST, que contiene la clase Server.

Figura 5-67. Clases implicadas para crear una ventana Server Event Logger.

En la clase Server se han creado, dos métodos especialmente para leer el Event Logger del servidor OPC. Uno denominado GetEventLoggerDefault, que no recibe ningún parámetro. Simplemente llamando este método nos devuelve una lista de 100 objetos de tipo EventLogger.

El otro método es más complejo ya que permite aplicar un filtro a los eventos a leer. Por lo tanto, recibe como parámetros la fecha y hora de inicio del filtro, como la fecha y hora del final y por último la cantidad máxima de eventos a consultar. Esta forma de filtrar los eventos ya está definida en la API REST del servidor. De forma que la URI de esta petición contiene los tres parámetros nombrados anteriormente. El método ya se encarga de devolver la respuesta del servidor REST, tras convertir el JSON en una lista de objetos.

Vistos los métodos, ahora se explicará como estos métodos son utilizados para programar los dos modos de funcionamiento de esta ventana.

Para que el usuario pueda hacer las consultas en modo automático (ver Figura 5-68), se ha creado un temporizador que se pone en marcha, cuando el usuario hace click sobre el check Auto. Este interrumpe el hilo principal de la clase, cuando ha pasado el tiempo x que ha configurado el usuario como intervalo de temporización. De esta forma en cada interrupción la clase Event Logger, se llama al método GetEventLoggerDefault, para consultar los últimos 100 eventos

registrados. Una vez el método devuelve la lista de eventos, estos se introducen en la tabla ya creada en esta ventana. Antes de introducir los datos se limpia la tabla para borrar los eventos mostrados anteriormente. De esta forma solamente se verán los últimos 100 eventos.

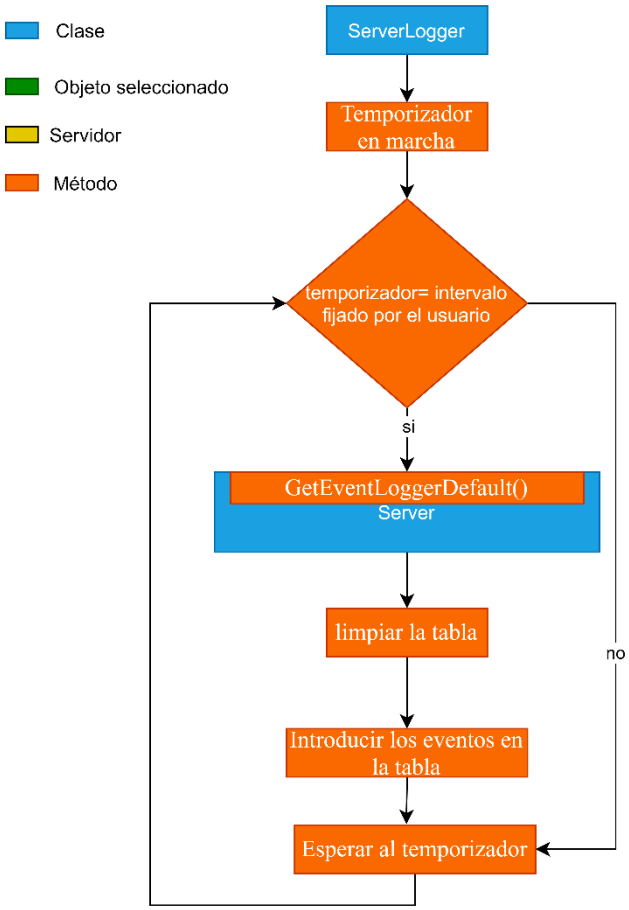


Figura 5-68. Diagrama de flujo que explica el proceso de lectura del Logger en modo automático.

Para el modo con Filtro, se han creado varios controles en la ventana (ver **Figura 5-69**) que permiten introducir las fechas y el límite de eventos a leer. También se ha creado un botón " GET LOGGER" mediante el cual el usuario lanza la petición para leer el Event Logger con el filtro aplicado. Esta petición es posible gracias al método GetEventLoggerWithFilter de la clase Server. Se lanza la petición de lectura llamando a este método, donde se pasan los parámetros determinados. Y este método devuelve una lista de eventos.

En este caso antes de introducir estos eventos en la tabla directamente, se pasan los eventos por otro filtro. Este filtro se ha diseñado en la propia aplicación de configuración, para que el usuario pueda filtrar la lista de eventos por el tipo de evento.

Para ello, se han creado 5 check box en esta ventana (ver **Figura 5-69**) . Que son el All, Security, Warning, Information y Error. Si el check All está marcado no se aplica ningún filtro y la lista entera será introducida y mostrada en la tabla. Pero en caso de elegir algún otro check o una combinación de ellos, mediante un bucle que recorre toda la lista, se filtrarán los eventos y solamente se mostrarán los eventos de acuerdo con los checks marcados.

Otra de las opciones que proporciona esta ventana secundaria, es la posibilidad de exportar el Event Logger en un archivo de texto. Para ello en la ventana se ha creado un control de tipo botón llamado " SAVE AS…". Si el usuario hace click sobre este botón, se abre una nueva ventana que pide al usuario el nombre del archivo y la ubicación de dónde se va a almacenar. Una vez seleccionada la ruta y el nombre, se procede a leer la tabla con todos los eventos que está mostrando. Cada iteración de este bucle lee y almacena en el archivo una fila de la tabla, donde a la hora de escribir en el archivo cada columna de la fila es separada por una "coma". Esta utilidad de exportar el Event Logger, no depende del modo de funcionamiento ya sea automático o filtrado.

A parte de decidir el modo de funcionamiento, el check "Auto", también tiene otra utilidad. Se ha programado de tal forma que dependiendo del modo que se ha seleccionado, se esconden o se muestran algunos de los controles. Es decir, para el modo automático, solamente se muestran dos controles. Uno que es el cuadro de texto numérico (sirve para que el usuario pueda introducir el intervalo de temporización) y otro es el botón "SAVE AS…" (ver **Figura 5-70**). En el modo filtrado se muestran todos los controles restantes (ver **Figura 5-69**).

Para lograr esta funcionalidad, de nuevo se juega con la visibilidad de estos controles.

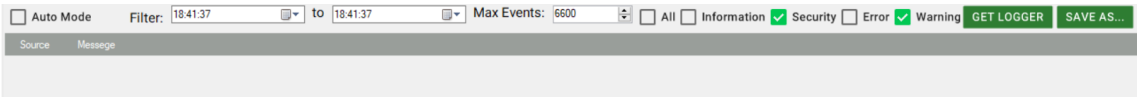


Figura 5-69. Figura que muestra los controles que son visibles en modo filtrado.

Figura 5-70. Figura que muestra los controles que son visibles en modo automático.

MultiLogger

Esta clase va asociada a la ventana MultiLogger, la cual sirve básicamente para mostrar dos ventanas secundarias de tipo Server Event Logger. En esta ventana se han agregado dos contenedores. Dentro de cada contenedor se ha agregado un check box "Enable" y una lista desplegable. Esta clase usa la misma idea que ya usamos en la clase Main para cargar la lista de las conexiones guardadas en el archivo servers.txt. Hay un temporizador que, de forma cíclica, va actualizando los datos tal y como ya vimos en la clase Main.

Una vez cargada la lista de todas las conexiones, se elige una conexión y se hace click sobre el check "Enable". Se crea una instancia de la clase ServerLogger, pasando la conexión seleccionada como parámetro.

OPCUAEndpoint

La ventana secundaria OPC UA Endpoints tiene asociada la clase OPCUAEndpoint. La misión de esta ventana es permitir al usuario consultar, crear, modificar y borrar los OPC UA Endpoints. Esta clase se instancia desde la clase principal (Main). El constructor de la clase recibe como parámetro una conexión, con la cual puede crear una instancia de la clase Server, para poder comunicar con el servidor OPC. Por eso es obligatorio que el usuario en la ventana principal esté conectado a un servidor. En caso de que el usuario no esté conectado a ninguna conexión, se

mostrará una ventana popup de advertencia (ver **Figura 5-71**), indicando que se debe conectar a un servidor.

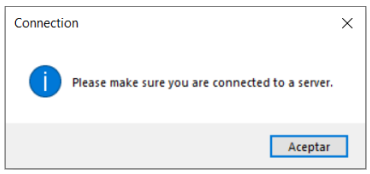


Figura 5-71. Ventana popup que se muestra cada vez que el usuario no está conectado a un servidor, pero quiere acceder a una funcionalidad que requiere una conexión con el servidor.

Cuando se abre la ventana OPC UA Endpoints (ver **Figura 5-72**), lo primero de todo se procede a consultar los OPC UA Endpoints existentes en el Servidor OPC. Para ello en la clase Server se ha creado un método llamado GetOPCUAEndpoints, que se encarga de consultar los Endpoints con el Servidor OPC y los devuelve en una lista de objetos de tipo UAEndpoint.

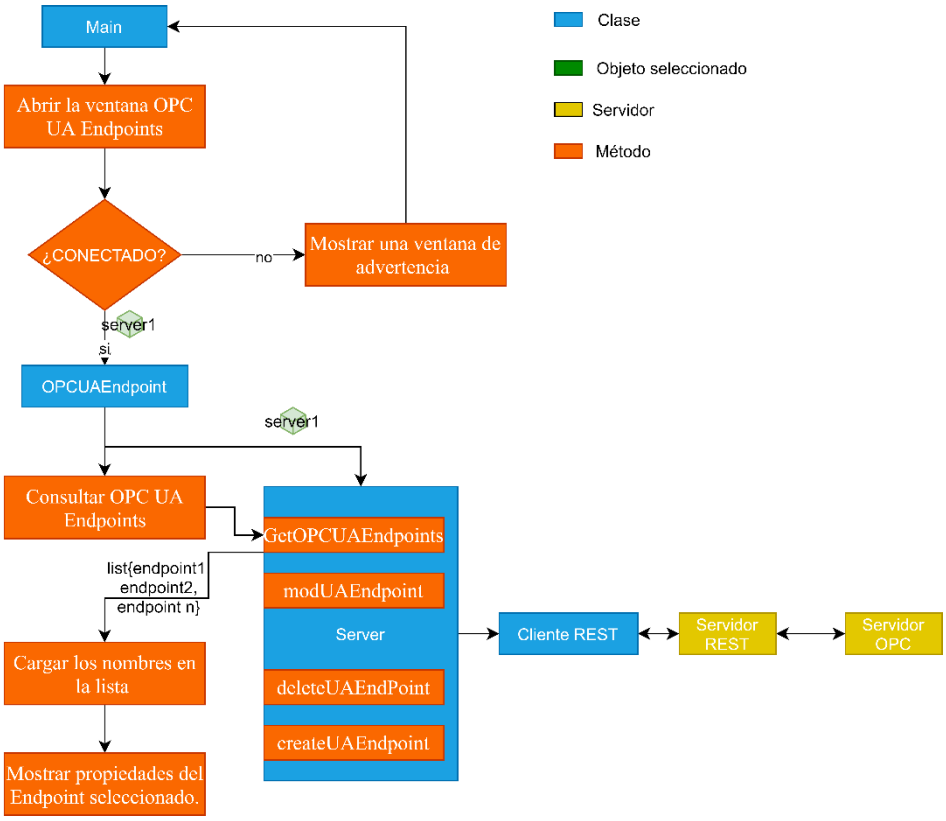


Figura 5-72. Diagrama de flujo de la ventana secundaria OPC UA Endpoints.

Se carga el nombre de cada OPC UA Endpoint recibido en una lista de selección, que ya está creada en la ventana OPC UA Endpoints, dentro del grupo Endpoints. Cuando se selecciona un OPC UA Endpoint desde esta lista, se cargan todas las propiedades restantes en la ventana, mediante varios cuadros de textos, desplegables y check boxs.

El Endpoint mostrado es totalmente configurable, pero para enviar estos cambios al servidor OPC, el usuario debe hacer click sobre el botón “SAVE CHANGES”. Tras hacer click sobre este botón, se vuelve a usar la instancia de la clase Server, llamando al método modUAEndpoint. Este método recibe el objeto UAEndpoint y el nombre actual del mismo. Gracias a estos dos parámetros este

método es capaz de enviar la petición adecuada al Servidor OPC para modificar el OPC UA Endpoint.

Para borrar un Endpoint, se usa la misma idea. El OPC UA Endpoint seleccionado es enviado al método correspondiente de la clase Server. Este ya se encarga de enviar la petición adecuada para borrar este OPC UA Endpoint.

Para implementar la funcionalidad de crear un nuevo OPC UA Endpoint, se ha tenido que enfrentar a un "problema" que quizá es un fallo en el diseño de la API REST.

A la hora de crear un OPC UA Endpoint el usuario debe seleccionar una de las tarjetas de red disponibles en la máquina donde está instalado el Servidor OPC.

Cada vez que se consulta un OPC UA Endpoint ya existente, en una de las propiedades llamada "DESCRIPTION" (ver **Figura 5-73**) viene una única cadena de texto, con todas las tarjetas de red disponibles.


```
"common.ALLTYPES_NAME": "local",
"common.ALLTYPES_DESCRIPTION": "Available adapters: Default; wlp2s0:192.168.1.123; enp1s0:<disconnected>; localhost",
"libadminsettings.UACONFIGMANAGER_ENDPOINT_ADAPTER": "localhost",
"libadminsettings.UACONFIGMANAGER_ENDPOINT_PORT": 49330,
"libadminsettings.UACONFIGMANAGER_ENDPOINT_URL": "opc.tcp://127.0.0.1:49330",
"libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_NONE": true,
"libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC128_RSA15": 0,
"libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC256": 0,
"libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC256_SHA256": 0
```


Figura 5-73. Respuesta del servidor REST cuando se consulta un OPC UA Endpoint.

Mediante código y lógica de programación se procesa esta cadena de texto para convertirla en una lista desplegable con todas las tarjetas de red disponibles (ver **Figura 5-74**). De esta forma ya tenemos todas las tarjetas de red disponibles en la aplicación de configuración. Usando esta información ya se puede crear un nuevo OPC UA Endpoint. Pero recordamos que todo esto es posible, porque hemos podido consultar esta información a partir de un OPC UA Endpoint ya creado.

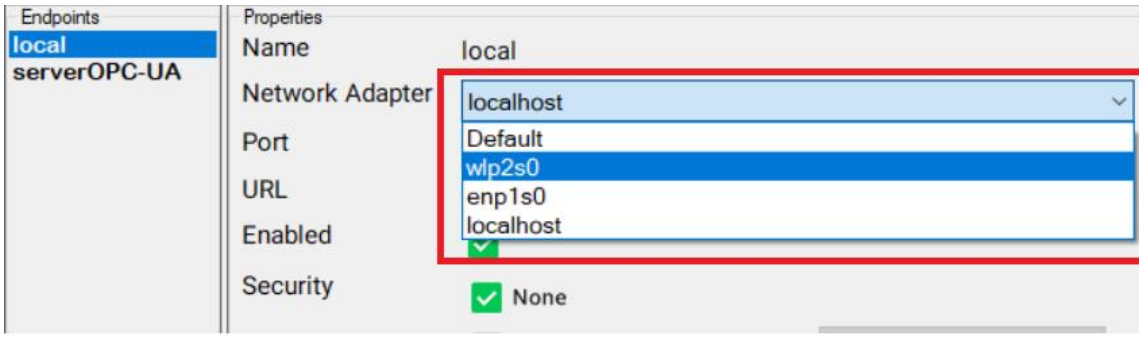


Figura 5-74. Lista de todas las tarjetas de red disponibles en la máquina con servidor OPC.

Pero supongamos que es la primera vez que se crea un Endpoint, es decir, actualmente el Servidor OPC no tiene ningún OPC UA Endpoint creado. En esta situación, la API REST no tiene ninguna función implementada para que se puedan consultar las tarjetas de red disponibles.

Para intentar "solucionar" este problema, ante esta situación donde no tenemos ninguna tarjeta de red podemos aprovechar dos opciones que seguramente son válidas. Localhost y Default siempre pueden ser aceptadas por el Servidor OPC. Ya que localhost siempre existe en todas las máquinas y default significa que el servidor OPC decide por su cuenta para elegir la tarjeta de red por defecto.

Finalmente, cuando ya tenemos las tarjetas de red, el usuario puede crear un nuevo Endpoint haciendo click sobre el botón "NEW". Cuando se hace click sobre este botón, se limpian todos los controles (cuadros de texto, desplegables, check boxs, etc.) por si tienen las propiedades de un OPC UA Endpoint ya cargado previamente. El botón "NEW" cambia de texto descriptivo de "NEW" a "CREATE". De esta forma cuando el usuario acaba de introducir las propiedades necesarias, puede hacer click sobre el botón "CREATE" para lanzar la petición de crear el OPC UA Endpoint al servidor REST.

Cuando se hace un click sobre el botón "NEW" aparte de limpiar los controles, también da un valor por defecto a todas las propiedades para que, cuando leamos estas propiedades, no haya ningún campo sin valor. Recordemos que en C# si se accede a un control que no tiene ningún valor esto genera una excepción.

Durante todo el proyecto se han dado valores a todos los campos de todas las ventanas para evitar este tipo de excepciones. Pero aun así el código escrito está dentro de un bloque try catch para capturar este tipo de excepciones.

Para crear el OPC UA Endpoints, se leen todas las propiedades y se guardan en un objeto de tipo UAEndpoint y se utiliza la instancia de la clase Server y su método createUAEndpoint.

Si el Endpoint se ha creado correctamente el método devuelve un true. En este caso se vuelve a llamar al método GetOPCUAEndpoints, para volver a consultar los Endpoints creados y se cargan a la lista.

About

La clase que se encarga de dar vida a la ventana secundaria llamada About. Esta clase no tiene ningún método implementado, ya que solamente se usa para mostrar información de contacto con un par de textos e imágenes que no realizan ninguna otra función.

GroupTagCreator

La clase que va asociada a la ventana secundaria New Tag Group Creator. Esta ventana solamente es accesible desde la sección New Configuration, dentro de la pestaña "NEW TAG". Cuya única misión es permitir al usuario crear un nuevo grupo de tags.

Para abrir esta ventana el usuario desde la pestaña "NEW TAG" debe seleccionar un canal y un dispositivo. Solamente después de seleccionar el canal y dispositivo se hace visible un botón denominado "NEW GROUP". Al hacer click sobre este botón, se hace una instancia de la clase GroupTagCreator. El constructor de la clase recibe el nombre del canal y dispositivos seleccionados, junto con el objeto ServerProperties (la conexión seleccionada).

A continuación, la clase GroupTagCreator se encarga de crear una instancia de la clase Server gracias al objeto ServerProperties. También se crea una instancia de la clase TagGroup, para que el usuario pueda asignar las propiedades correspondientes para crear un grupo de tags.

Hay que recordar que la clase TagGroup es una clase simple que contiene las propiedades de un grupo de tags.

En esta ventana secundaria solamente se han creado dos cuadros de textos, ya que un grupo de tags solamente requiere un nombre y una breve descripción. El nombre es obligatorio, pero el campo de descripción es opcional.

Una vez introducidos estos valores y que el usuario hace click sobre el botón "CREATE" (ver **Figura 5-75**), se procede a leer y guardar el nombre y la descripción del grupo de tag, en el objeto TagGroup. Seguidamente, este objeto junto con el nombre del canal y dispositivo son enviados a la clase Server, llamando al método creatNewTagGroup. Este método se encarga de enviar la petición adecuada construyendo la URI y body necesario. Si el grupo se ha creado correctamente este método devuelve un true.

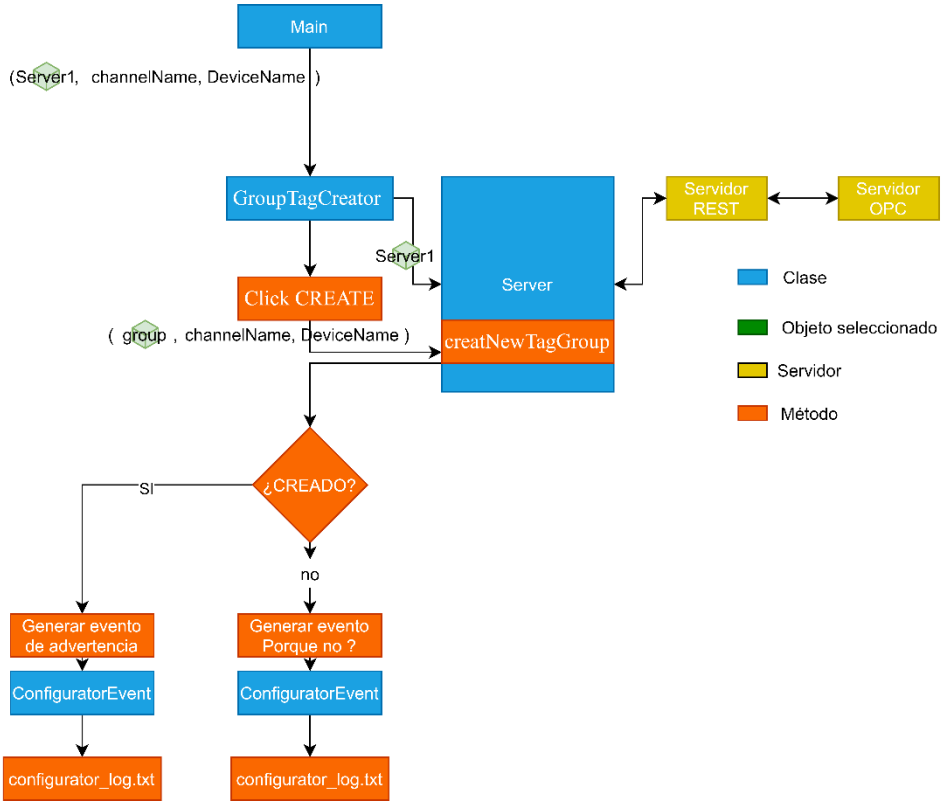


Figura 5-75. Diagrama de flujo que explica cómo se crea un nuevo grupo de tags.

En caso de que el grupo de tags se haya creado correctamente, la clase GroupTagCreator registra un evento de tipo ConfiguratorEvent (ver **Figura 5-76**), para indicar al usuario que el grupo se ha creado correctamente. En caso contrario, cualquier problema "excepción" también será registrada para que el usuario sepa cuál es el motivo por el cual el grupo no fue creado correctamente.

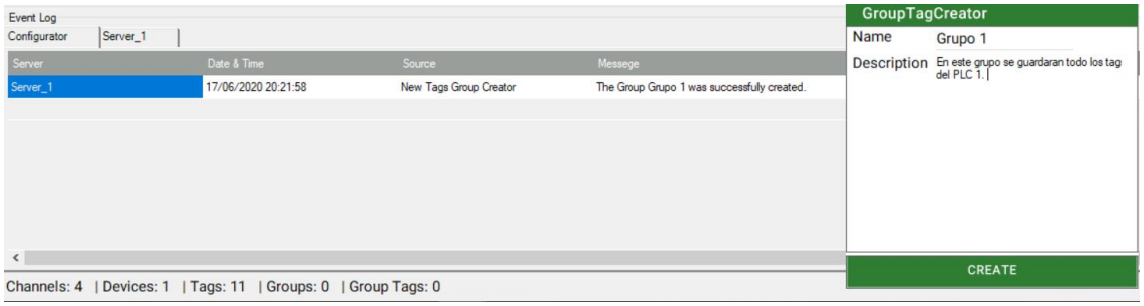


Figura 5-76. El evento registrado cuando un grupo se ha creado correctamente.

Finalmente, una vez acabado de programar la aplicación de configuración, desde el departamento de Solution Managers de Logitek, se ha decidido poner KEPServerEX RTC como el nombre de la aplicación. Donde las siglas RTC significan Remote Tool Configurator.

El logo de la aplicación (ver **Figura 5-77**) fue diseñado desde el departamento comercial.

Figura 5-77. Logo de la aplicación KEPServerEX RTC.

Con una aplicación ya programada y 100% funcional, es hora de crear un asistente de instalación. Hasta ahora la aplicación siempre se había ejecutado desde la IDE de Visual Studio.

La carpeta del proyecto de Visual Studio ya contiene un archivo .exe de nuestra aplicación. Copiando esta carpeta de proyecto en cualquier otro ordenador con sistema operativo de Windows se podría ejecutar la aplicación. Pero en este caso para hacerlo de una forma más profesional se ha decidido crear un asistente de instalación.

5.4.4. Asistente para instalación

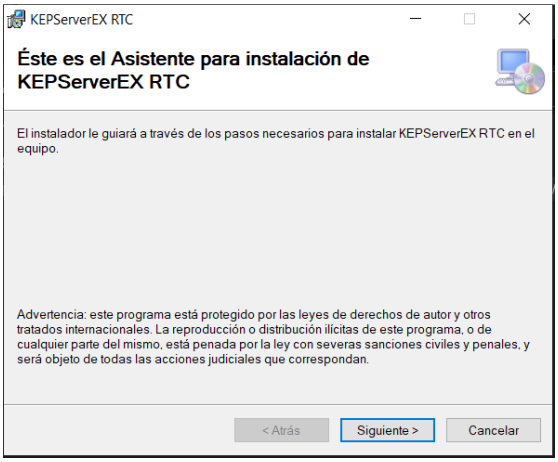


Figura 5-78. Asistente para instalar KEPServerEX RTC.

Desde el administrador de extensiones de Visual se instala un paquete de librería que ofrece Microsoft de forma gratuita.

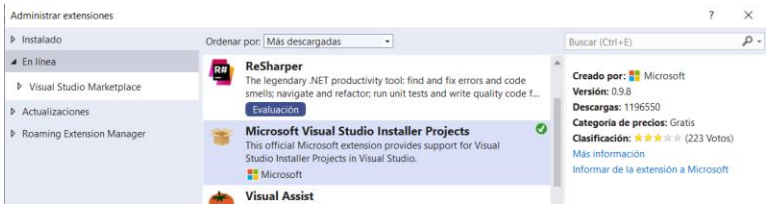


Figura 5-79. Administrador de extensiones de Visual Studio.

La extensión en concreto a descargar es Microsoft Visual Studio Installer Projects (ver **Figura 5-79**). Esta herramienta permite crear un instalador de nuestro proyecto de una forma muy visual sin tener que escribir nada de código.

Con esta herramienta se ha creado un instalador de nuestro proyecto con las siguientes configuraciones:

- La ubicación de instalación será C:\Program Files (x86)\Logitek S.A\KEPServerEX RTC (ver **Figura 5-80**).
- Una vez instalada la aplicación se crearán tres accesos directos. Uno en el escritorio, otro en la barra de búsqueda y otro que estará dentro de la carpeta donde se instaló la aplicación.
- El asistente detecta si es una versión de aplicación superior o inferior si este se ejecuta en una máquina que ya tiene instalada la aplicación.
- Detecta si no está instalado Microsoft .NET Framework. En caso de que no esté, da la posibilidad de descargarlo e instalarlo.

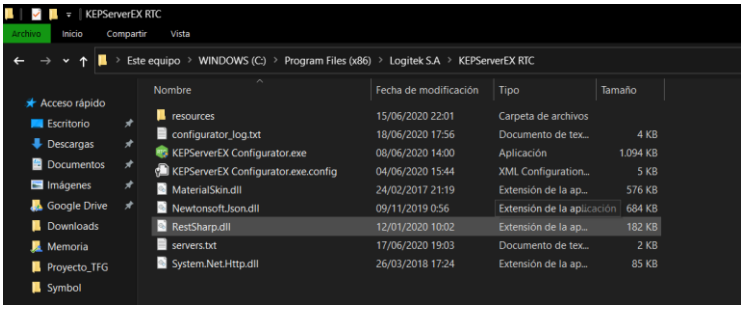


Figura 5-80. Ubicación y contenido de la carpeta donde se instala la aplicación.

5.4.5. Testing de Software.

Una vez comprimido el proyecto en un instalador, se ha procedido realizar varias pruebas para comprobar la calidad de la aplicación.

Primero de todo se ha instalado en diferentes ordenadores con diferentes sistemas operativos para comprobar que la aplicación se puede instalar y ejecutar sin problemas.

Se ha probado en versiones de Windows:

- Windows 10

- Windows Server 2012
- Windows Server 2016
- Windows 8

Después de instalar la aplicación, se realizaron pruebas de comunicación con el servidor OPC que está instalado en el ordenador con el sistema Linux.

Una vez instalada la aplicación, se han realizado pruebas utilizando todas las funciones implementadas en la aplicación para buscar los posibles bugs. Como era de esperar la aplicación tenía varios bugs, que se han ido resolviendo a medida que se iban detectando.

Se ha creado una lista (recopilatorio) de los bugs más importantes que se han detectado:

- Si **no** se introducen todos los valores a la hora de crear un canal, el bucle que leen todos los campos, si se enfrenta con un campo que está vacío, se produce una excepción que hace que se cierre la aplicación. Para solucionar este problema se incluye un bloque try catch para recoger estas excepciones. También se modifica el código para que detecte esta situación y no lea campos que están vacíos.
- Cuando el usuario está conectado a un servidor y desde la ventana Server Connections Manager borra todas las conexiones guardadas, en la ventana principal aún se seguía mostrando que estaba conectado a un servidor. Para ello se detecta esta situación y dentro de la ventana principal se desconecta la conexión actual. También se limpia toda la sección Current Configuration para que no muestre la información si se está desconectado.
- Se corrige el formato del timestap (ver **Figura 5-81** y **Figura 5-82**) de un evento registrado por el servidor OPC. La fecha y hora devuelta por el servidor REST, es una forma conjunta de fecha y hora con un señalador de hora 'T'. Además, la hora está atrasada dos horas que la hora de la máquina donde está instalado el servidor OPC.

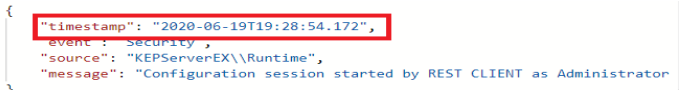


Figura 5-81. Timestap que envía el servidor OPC.

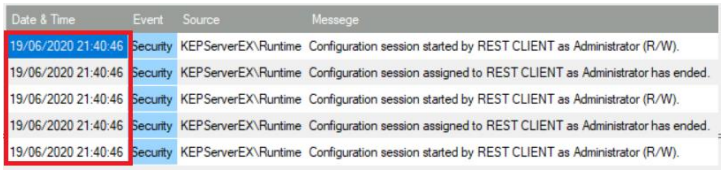


Figura 5-82. Formato de timestap corregido.

Dado KEPServerEX RTC también permite configurar KEPServerEX (versión de servidor OPC para Windows), también se han realizado varias pruebas para configurar dicha versión. Es evidente que nuestra aplicación tiene ciertas limitaciones para este Servidor OPC, ya que solamente se ha programado para configurar 4 tipos de drivers. KEPServerEX dispone de muchos más. Además, la API REST de KEPServerEX no dispone la funcionalidad de configurar los OPC UA Endpoints. Excepto estos detalles todas las pruebas se realizaron exitosamente.

Tras corregir varios bugs y estar seguro de que la aplicación ya permitiera configurar e interactuar sin dar problemas, se decide enviar la versión beta a los ingenieros de varios departamentos de Logitek, para que lo prueben y hagan pruebas de comunicación y configuración.

Antes de enviar la versión beta, se ha creado una guía (ver Anexo III), que explica de forma breve como hay que utilizar KEPServerEX RTC para configurar el servidor OPC.

Todo el feedback recibido de todas las personas que han probado la aplicación se recopila en una lista, para seguir mejorando la aplicación. Esta forma de testear la aplicación ha servido mucho para mejorar la interfaz gráfica de la aplicación, realizando varias interacciones.

Por ejemplo, en la primera iteración (ver **Figura 5-83**) se aplicó el concepto de interfaz gráfica con material design para cambiar el estilo de los formularios y los controles que vienen por defecto. En esta versión también se introduce el concepto de encapsular las propiedades de un server en un único concepto llamado conexión.

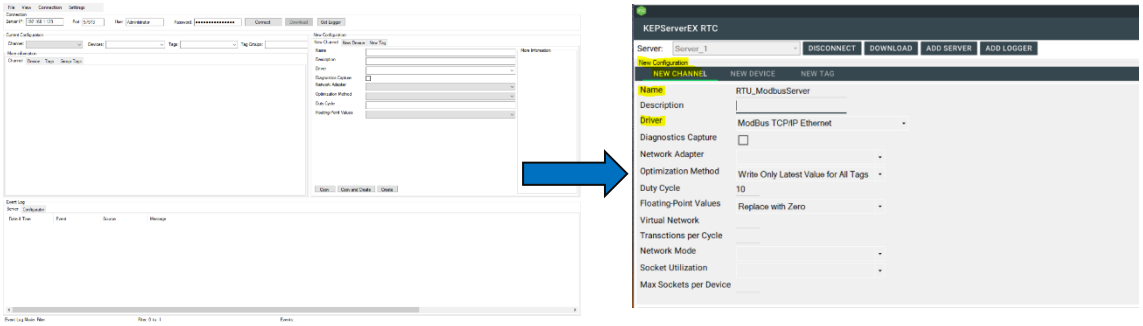


Figura 5-83. Primera iteración para mejorar el diseño de la interfaz gráfica.

En la segunda iteración (ver **Figura 5-84**) basándose en el feedback recibido se introducen varias mejoras en el diseño, principalmente para mejorar la visualización de la información, cambiando los colores de los controles, color de fondo de las ventanas, tipo y tamaño de letra de los botones y cuadros de texto.

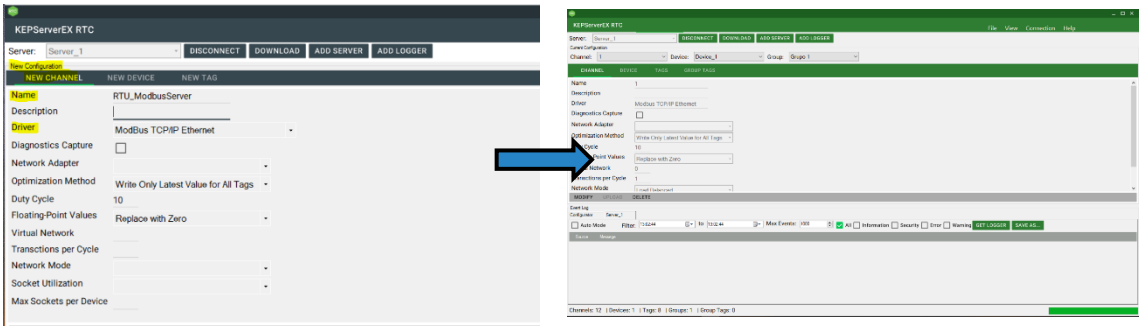


Figura 5-84. Segunda iteración para mejorar el diseño de la interfaz gráfica.

6. Aplicación SCADA

Un SCADA (Supervisory Control And Data Acquisition) es una aplicación que permite controlar y supervisar procesos industriales. Permite leer y escribir datos provenientes de los dispositivos de campo. Los SCADA modernos de hoy en día a parte de supervisar un proceso, también permiten almacenar datos en una base de datos, generar informes, integración con programas de mantenimiento de los equipos, gestión de alarmas, etc.

El objetivo de crear una aplicación SCADA, es para demostrar el correcto funcionamiento de la aplicación de configuración. La idea es aprovechar la arquitectura montada para la realización de las pruebas (ver **Pruebas de configuración del servidor OPC vía API REST**), pero en este caso el cliente OPC UA será nuestro SCADA. De manera que los datos provenientes de la RTU se mostrarán en la aplicación SCADA.

En el mercado existen muchos productos que permiten crear aplicaciones SCADA. Cada uno con sus ventajas y desventajas. Dado que este proyecto se está realizando bajo la supervisión de la empresa Logitek S.A, por lo tanto, en este caso se ha decidido utilizar Intouch Edge, porque Logitek S.A es el distribuidor oficial de este producto a nivel de España y Portugal.

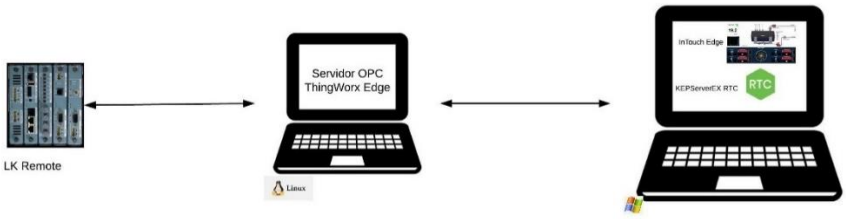


Figura 6-1. Arquitectura para lograr la comunicación entre la RTU y SCADA.

La RTU intercambiará los datos con el servidor OPC (ver **Figura 6-1**), que en este caso está instalado en un ordenador con el sistema operativo Linux. El servidor se configurará con la aplicación KEPServerEX RTC que estará instalado junto con el SCADA, en el segundo ordenador con sistema operativo Windows.

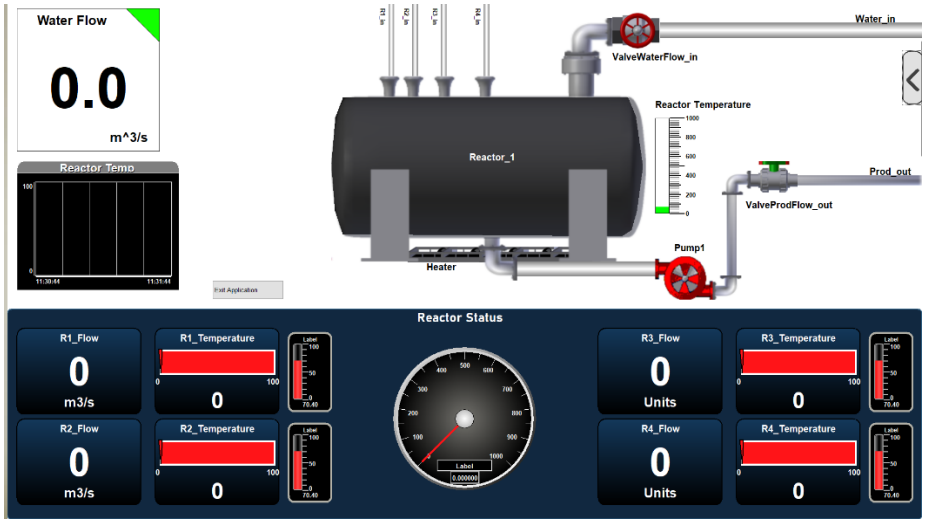


Figura 6-2. Aplicación SCADA

La aplicación SCADA (ver **Figura 6-2**) desarrollada en este caso es un proceso industrial simulado, donde se controlará un reactor.

Los elementos para supervisar son:

- La temperatura y caudal de los cuatros reactivos de entrada.
- Temperatura del reactor.
- Nivel del reactor
- El caudal de entrada de agua.
- Válvula para controlar el caudal de agua.
- Calentadores.
- Bomba de salida
- Válvula de salida

Algunos de estos datos provienen de la RTU y otros son simulados utilizando el driver simulador del servidor OPC. La RTU se ha programado utilizando el programa TwinSoft. Pero para los tags simulados por el propio servidor, se ha utilizado el KEPServerEX RTC (ver **Figura 6-3**).

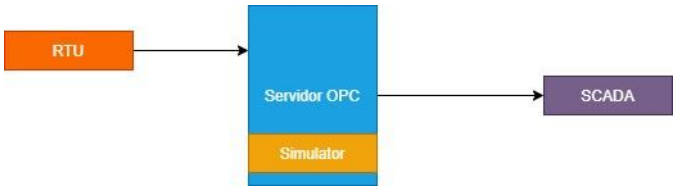


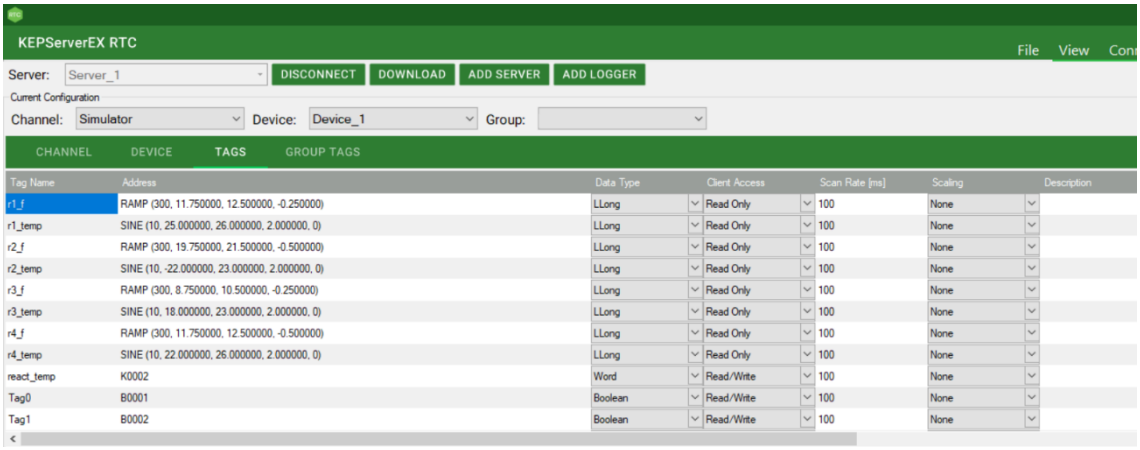
Figura 6-3. Obtención de los datos.

Para configurar la RTU en el servidor OPC, se ha aprovechado la misma configuración que ya se había realizado durante la fase de pruebas. Lo único que cambia es la cantidad de tags. Para ello se ha utilizado la aplicación KEPServerEX RTC para agregar los nuevos tags programados en la RTU.

Seguidamente, se configura el servidor OPC para crear un nuevo canal utilizando el driver Simulador.

Dentro de ese canal se crea un dispositivo nuevo, dejando todos los valores por defecto. Acto seguido se crean los tags. El driver simulador permite crear una gran variedad de variables, ya sean para simular datos booleanos, constantes, rampas, ondas sinusoidales, etc.

Finalmente se crean los tags (booleanos, constantes, rampas y ondas sinusoidales) necesarios para simular nuestro proceso industrial (ver Figura 6-4).



Tag Name	Address	Data Type	Client Access	Scan Rate [ms]	Scaling	Description
r1_f	RAMP (300, 11.750000, 12.500000, -0.250000)	LLong	Read Only	100	None	
r1_temp	SINE (10, 25.000000, 26.000000, 2.000000, 0)	LLong	Read Only	100	None	
r2_f	RAMP (300, 19.750000, 21.500000, -0.500000)	LLong	Read Only	100	None	
r2_temp	SINE (10, -22.000000, 23.000000, 2.000000, 0)	LLong	Read Only	100	None	
r3_f	RAMP (300, 8.750000, 10.500000, -0.250000)	LLong	Read Only	100	None	
r3_temp	SINE (10, 18.000000, 23.000000, 2.000000, 0)	LLong	Read Only	100	None	
r4_f	RAMP (300, 11.750000, 12.500000, -0.500000)	LLong	Read Only	100	None	
r4_temp	SINE (10, 22.000000, 26.000000, 2.000000, 0)	LLong	Read Only	100	None	
react_temp	K0002	Word	Read/Write	100	None	
Tag0	B0001	Boolean	Read/Write	100	None	
Tag1	B0002	Boolean	Read/Write	100	None	

Figura 6-4. Captura de pantalla que muestra la configuración realizada para el driver simulador.

El motivo por el cual se ha decidido utilizar el driver simulador, es para demostrar la funcionalidad de modificar datos desde la aplicación de configuración. Es decir, supongamos que actualmente la temperatura del reactor es un tag simulado, que va desde 100 a 200 grados centígrados. Una vez el SCADA ya está mostrando los datos, podemos utilizar la aplicación de configuración para modificar esta variable para que ahora vaya de 150 a 200 grados. O podemos borrar un tag en el servidor OPC, de esta forma el SCADA debería mostrar un error, indicando que no es capaz de leer este tag borrado.

Tras configurar el servidor OPC, en el SCADA Intouch Edge se configura un cliente OPC UA. Este cliente OPC UA apunta al servidor OPC instalado en la máquina con sistema operativo Linux. El explorador del cliente OPC UA (ver Figura 6-5) muestra todos los canales, dispositivos y tags creados. Se agregan los tags simulados y los provenientes de la RTU y para mostrar estos datos, se vinculan a las variables a los símbolos ya creados en el SCADA.

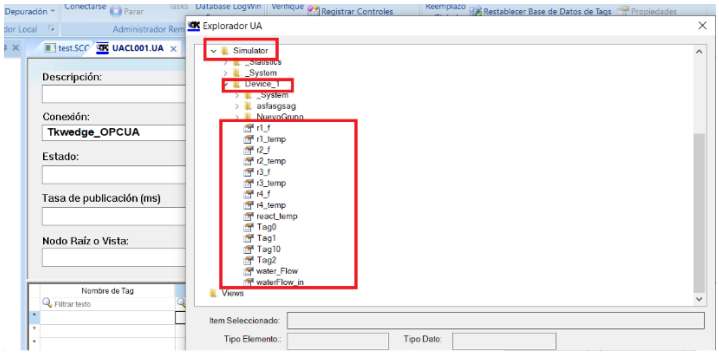


Figura 6-5. Captura que muestra el explorador del cliente OPC UA, que muestra todos los tags del canal simulador.

Finalmente, se ejecuta la aplicación SCADA y se observa que ya se están mostrando todos los tags correctamente (ver Figura 6-6).

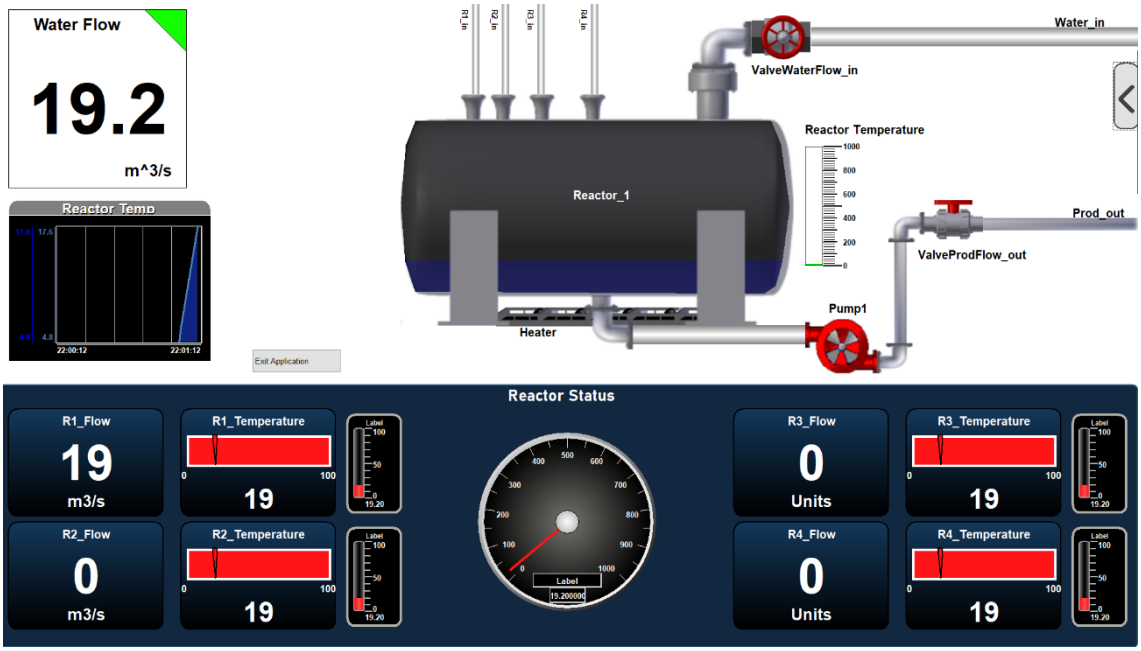


Figura 6-6. Aplicación SCADA mostrando los datos.

7. Conclusiones

Tras la finalización del proyecto se ha llegado a las siguientes conclusiones:

- La API REST es una utilidad de gran valor añadida a las plataformas de comunicaciones industriales de Kepware, que permite configurar el servidor OPC de forma remota. Lo cual puede ser muy ventajoso para empresas que tienen desplegados muchos servidores OPC.

- Con la documentación que ha proporcionado el fabricante, es fácil aprender y comprender el funcionamiento de la API REST. Pero también cabe destacar que esta utilidad requiere unos conocimientos extras de las peticiones HTTP y la estructura de texto en formato JSON.

- Con las primeras pruebas de configuración se concluye que, para cada petición se debe crear una estructura de texto en formato JSON. Esto puede llegar a ser muy tedioso si se tiene que crear una gran cantidad de canales, dispositivos o tags. A su vez cualquier error en la sintaxis de la estructura JSON, puede provocar la cancelación de una petición.

  La respuesta del servidor también está en forma JSON y devuelve todos los parámetros del objeto consultado, el cual dificulta la lectura de la información de más interés.

  Definitivamente configurar el servidor OPC utilizando la API REST no es una forma ágil y práctica para un usuario.

- ThingWorx Kepware Edge funciona tal y como se esperaba. Permite comunicar con los dispositivos de campo para la obtención de los datos y finalmente sirve estos datos a sistemas de nivel superior a la perfección.

- Antes de diseñar la aplicación de configuración final, se planteó crear una aplicación "sencilla" de pruebas para comprobar, que realmente la elección de utilizar Visual Studio como IDE y el lenguaje de programación C#, cumplirán con los requisitos mínimos que debe de cumplir la aplicación de configuración final.

  Esta decisión fue un éxito ya que, con la aplicación de pruebas desarrollada, demostró que tanto Visual Studio como C#, están a la altura para cumplir con todos los requisitos planteados.

- El diseño de la interfaz gráfica para la aplicación final ha sido una tarea "sencilla", gracias a los requisitos planteados y teniendo en cuenta las peticiones de mejoras recibidas para el KEPServerEX, en la aplicación de configuración, se han podido incluir funcionalidades que no tenía la interfaz gráfica de KEPServerEX.

- La programación de la aplicación KEPServerEX RTC ha sido una de las fases más complejas de todo el proyecto. Se ha tenido que hacer frente a varios problemas que resolver para lograr los objetivos planteados. Sin tener conocimientos previos de

programación con C#, es todo un reto construir una aplicación de esta dimensión en tan solo 3 meses de desarrollo. Pero gracias al curso online sobre programación orientada a objetos con C#, la documentación proporcionada por Microsoft, pero sobre todo la ayuda de los foros (Stack Overflow, Microsoft, lawebdelprogramador, etc.) de grandes comunidades de programadores que hay en internet, se han logrado todos los objetivos marcados para la aplicación de configuración.

- Gracias a la fase de Testing y el feedback recibido por todas las personas que han participado en esta fase, se ha logrado una aplicación 100 % funcional que permite configurar el servidor OPC de una forma fácil, rápida y eficaz.

- La aplicación SCADA desarrollada, ha servido para demostrar el correcto funcionamiento de KEPServerEX RTC. Durante esta prueba se ha visto como los datos provenientes de la RTU y los tags simulados por el propio servidor OPC, son mostrados correctamente en la aplicación SCADA.

- Sobre todo, este proyecto ha servido para ver el rumbo actual de la industria, las nuevas tecnologías basadas en la industria 4.0 que se están desarrollando para conseguir unas fábricas más inteligentes y eficaces.

  Se han visto las ventajas que aporta la utilización de un servidor OPC como tecnología para la interoperabilidad.

  Se ha comprendido el funcionamiento de la API REST y las utilidades que puede traer esta tecnología en el ámbito de la automatización industrial.

  Gracias a la programación C# y la utilización de un entorno de programación como Visual Studio, se han ampliado los conocimientos de programación adquiridos durante la carrera.

8. ¿Cuál es el futuro de la aplicación de configuración?

La idea es llegar a conseguir una aplicación 100 % funcional que permita configurar no solamente el ThingWorx Kepware Edge sino también el KEPServerEX. Antes de diseñar la estructura principal de la aplicación, ya se había tenido en cuenta para que la aplicación sea fácil de escalar , para que se puedan añadir todos los drivers que tiene KEPServerEX.

De esta forma tan solo pequeñas modificaciones en el código y creando las clases simples necesarias, se pueden añadir los drivers necesarios.

Por otro lado, todas las otras funcionalidades que no se han podido añadir por falta de tiempo y toda la información recopilada gracias al feedback de las personas que han probado la aplicación, en un futuro se espera seguir trabajando para añadir las funciones como:

- Incluir los archivos .chm (ver **Figura 8-1**) que tiene cualquier típico programa, para proporcionar toda la ayuda necesaria dentro de la aplicación.

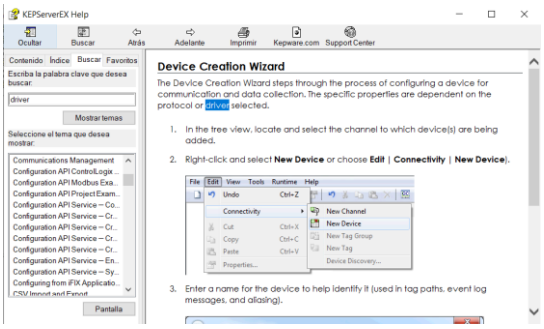


Figura 8-1. Ejemplo de una ventana típica de un archivo .chm.

- Actualmente la aplicación no permite cambiar un proyecto por otro que ya está creado y guardado previamente. Para ello se espera incluir una ventana secundaria para ver las propiedades de un proyecto cargado al Runtime del servidor OPC. Es decir, ver el nombre y la versión del proyecto, permitir al usuario habilitar o deshabilitar una interfaz (habilitar servidor OPC UA, en caso de KEPServerEX también OPC DA), habilitar o deshabilitar la captura de las tramas OPC UA/DA, etc.
- Importar y exportar proyectos en archivos en formato JSON.
- Incluir la posibilidad de gestionar los usuarios. Para ello se creará una nueva ventana secundaria.
- Incluir un botón denominado “CLEAR” que serviría para limpiar los campos de la sección New Configuration. Esto es para que, si el usuario ha introducido unos valores para crear un nuevo canal, pero a posteriori ya no quiere crear este canal y desea volver a empezar de cero.
- Habilidad de invertir los cambios. Esto sería posible si antes de enviar una petición, ya sea para crear, modificar o borrar un objeto, se realiza una copia de seguridad de la configuración actual, ya sea de forma temporal, en instancias de los objetos o directamente guardar esta información en un archivo “temporal”.

- Mediante el instalador (el asistente de instalación) permitir al usuario la posibilidad de elegir los drivers (las clases creadas) para solamente tener las funcionalidades que desea, de esta forma no tener que instalar toda la aplicación que obviamente ocuparía más.
- Investigar para implementación de un sistema de licencias para KEPServerEX RTC. Antes de comercializar un software que no sea de código abierto, es obvio que requiere un método de licenciamiento, que sea fiable y difícil de "piratear".

9. Presupuesto

Este apartado tiene como objetivo explicar de forma breve el presupuesto del proyecto realizado. El presupuesto se ha dividido en dos partes que son, el coste del material empleado y el coste de ejecución y la documentación del proyecto.

El precio €/hora es variable ya que se ha tenido en cuenta el tipo de actividad realizada. Las actividades se dividen en 3 tipos que son, realización de la documentación, preparación de los equipos y la programación. La mayor parte del tiempo se ha dedicado a la programación de la aplicación de configuración.

A continuación, se muestra el presupuesto total del proyecto (ver Tabla 9-1) . El presupuesto completo se detalla en el documento aparte denominado BUDGET.

Tabla 9-1. Coste total del proyecto.

Actividad	Total [€]
Coste del material empleado.	2994,95
Coste de realización del proyecto y la redacción de la documentación.	7284
Total	**10278,95**

10. Bibliografía

[1] T. Support, "Guía para configurar ThingWorx Kepware Edge utilizando KEPServerEX," 2020.

[2] "How to use Material Design Controls with C# in your WinForms application | Our Code World." https://ourcodeworld.com/articles/read/441/how-to-use-material-design-controls-with-c-in-your-winforms-application (accessed Jun. 01, 2020).

[3] "Serialization Attributes." https://www.newtonsoft.com/json/help/html/SerializationAttributes.htm (accessed May 25, 2020).

[4] "Qué es KEPServerEX, cómo funciona y qué te puede aportar." https://www.kepserverexopc.com/que-es-kepserverex-como-funciona-y-que-te-puede-aportar/ (accessed May 04, 2020).

[5] "Comparta los datos de su PLC con toda la empresa con OPC." https://www.matrikonopc.es/newsletter/content/connections/2012-11-hottopic.aspx?utm_campaign=Blast&utm_medium=Email&utm_source=MatrikonOPC&utm_content=ES-Connections-HotTopic-OPCServers-11-06-2012 (accessed May 03, 2020).

[6] "Products | Drivers | Suites | Plug-Ins | Kepware." https://www.kepware.com/en-us/products/thingworx-kepware-edge/ (accessed Jun. 23, 2020).

[7] "RestSharp." https://restsharp.dev/ (accessed Mar. 07, 2020).

[8] "Procedimiento Escribir en un archivo de texto: Guía de programación de C# | Microsoft Docs." https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/file-system/how-to-write-to-a-text-file# (accessed Apr. 17, 2020).

[9] "(1) ¿Que es un API REST? ¿Para qué sirve un API RESTful? ? ¿El futuro del Backend? Víctor Robles - YouTube." https://www.youtube.com/watch?v=4fZKFQjkNw0.

[10] "▷ ¿Qué es una API REST? | Guía 【2020】 - Idento." https://www.idento.es/blog/desarrollo-web/que-es-una-api-rest/ (accessed Jun. 15, 2020).

[11] "constructor - What is the best way to give a C# auto-property an initial value? - Stack Overflow." https://stackoverflow.com/questions/40730/what-is-the-best-way-to-give-a-c-sharp-auto-property-an-initial-value (accessed Apr. 23, 2020).

[12] "Kepware | Software for Industrial Automation and IoT." https://www.kepware.com/en-us/ (accessed Jun. 23, 2020).

[13] "The leading operating system for PCs, IoT devices, servers and the cloud | Ubuntu." https://ubuntu.com/#download (accessed Feb. 07, 2020).

[14] "Installation/FromUSBStick - Community Help Wiki." https://help.ubuntu.com/community/Installation/FromUSBStick (accessed Feb. 07, 2020).

[15] "Welcome to JSON.com!" http://json.com/ (accessed Mar. 20, 2020).

[16] "Maestro en C# en tiempo récord | Udemy." https://www.udemy.com/course/c-la-guia-definitiva-aprende-a-programar-de-cero/learn/lecture/15135550?src=sac&kw=c%23#overview (accessed Mar. 05, 2020).

[17] "Documentación de .NET | Microsoft Docs." https://docs.microsoft.com/es-es/dotnet/ (accessed Mar. 07, 2020).

[18] "Hola mundo: tutorial interactivo de introducción a C# | Microsoft Docs." https://docs.microsoft.com/es-es/dotnet/csharp/tutorials/intro-to-csharp/hello-world (accessed Mar. 07, 2020).

[19] "Descargar Visual Studio 2019 para Windows y Mac." https://visualstudio.microsoft.com/es/downloads/ (accessed Feb. 28, 2020).

[20] "Información general sobre Visual Studio | Microsoft Docs." https://docs.microsoft.com/es-es/visualstudio/get-started/visual-studio-ide?view=vs-2019 (accessed Feb. 28, 2020).

[21] "Support ThingWorx Kepware Edge | My Kepware." https://my.kepware.com/s/login/?ec=302&startURL=%2Fs%2F (accessed Jan. 27, 2020).

[22] "Create a bootable USB stick on Windows | Ubuntu." https://ubuntu.com/tutorials/tutorial-create-a-usb-stick-on-windows#1-overview (accessed Feb. 07, 2020).

[23] "Instantly parse JSON in any language | quicktype." https://app.quicktype.io/ (accessed Feb. 20, 2020).

[24] "Json.NET - Newtonsoft." https://www.newtonsoft.com/json (accessed Feb. 15, 2020).

[25] "Artículos Opc Server UA KEPServerEX - KEPServerEx." https://www.kepserverexopc.com/articulos-opc-server-ua-kepserver/ (accessed Jun. 02, 2020).

[26] "What is OPC? - OPC Foundation." https://opcfoundation.org/about/what-is-opc/ (accessed Jan. 02, 2020).

[27] "TWinSoft 11.06 SP3-Release notes All new features and corrections require the highest version of OS to run in your CPU. When receiving a TBox CPU, check its current version using TWinSoft doing a: 'Communication' → ' RTU Identification,'" 2016. Accessed: Jun. 23, 2020. [Online]. Available: http://download.microsoft.com/download/1/6/5/165255E7-1014-4D0A-B094-B6A430A6BFFC/vcredist_x86.exe.

[28] "La Web del Programador." https://www.lawebdelprogramador.com/ (accessed May 14, 2020).

[29] "Newest 'c#' Questions - Stack Overflow." https://stackoverflow.com/questions/tagged/c%23 (accessed May 11, 2020).

[30] "Information & Support - Unified Automation." https://www.unified-automation.com/support.html (accessed Feb. 04, 2020).

[31] Postman Inc, "How to Use Postman API Client: GraphQL, REST, & SOAP Supported," 2019. https://www.postman.com/product/api-client/ (accessed Feb. 24, 2020).

[32] "Anexo1_Manual ThingWorx Kepware Edge," PTC Inc, vol. 1, no. 2. p. 169, 2020.

[33] J. L. del Val Román, "Industria 4.0. La Transformación Digital de la Industria Española," Coddiinforme, p. 120, 2012, [Online]. Available: http://coddii.org/wp-content/uploads/2016/10/Informe-CODDII-Industria-4.0.pdf.

[34] "Plataformas de comunicación industrial basadas en tecnologías OPC" https://www.wonderware.es/servicios/formacion/plataformas-de-comunicacion-industrial-basadas-en-tecnologias-opc/